Modules over monads and operational semantics (expanded version)

André Hirschowitz, Tom Hirschowitz, Tom Hirschowitz

▶ **To cite this version:**

André Hirschowitz, Tom Hirschowitz, Tom Hirschowitz. Modules over monads and operational semantics (expanded version). 2022. hal-03447952v1

HAL Id: hal-03447952

https://hal.science/hal-03447952v1

Preprint submitted on 10 Feb 2022 (v1), last revised 15 Jun 2022 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

MODULES OVER MONADS AND OPERATIONAL SEMANTICS

ANDRÉ HIRSCHOWITZ, TOM HIRSCHOWITZ, AND AMBROISE LAFONT

Univ. Côte d'Azur, CNRS, LJAD, 06103, France

Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000, Chambéry,France

UNSW, Sydney, Australia

Abstract. This paper is a contribution to the search for efficient and high-level mathematical tools to specify and reason about (abstract) programming languages or calculi. Generalising the reduction monads of Ahrens et al., we introduce transition monads, thus covering new applications such as $\overline{\lambda}\mu$-calculus, π-calculus, Positive GSOS specifications, differential λ-calculus, and the simply-typed, call-by-value λ-calculus. Moreover, we design a suitable notion of signature for transition monads.

1. Introduction

The search for a mathematical notion of programming language goes back at least to Turi and Plotkin [60], who coined the name "Mathematical Operational Semantics", and explained how known classes of well-behaved rules for structural operational semantics, such as GSOS [19], can be categorically understood and specified via bialgebras and distributive laws. Their initial framework did not cover variable binding, and several authors have proposed variants which do [32, 31, 59], treating examples like the π-calculus. However, none of these approaches covers higher-order languages like the λ-calculus.

In recent work, following previous work on modules over monads for syntax with binding [42, 8] (see also [6]), Ahrens et al. [9] introduce **reduction monads**, and show how they cover several standard variants of the λ-calculus. Furthermore, as expected in similar contexts, they propose a mechanism for specifying reduction monads by suitable signatures.

Our starting point is the fact that already the call-by-value λ-calculus does not form a reduction monad. Indeed, in this calculus, variables are placeholders for values but not for general λ-terms; in other words, reduction, although it involves general terms, is stable under substitution by values only. In the present work, we generalise reduction monads to what we call **transition monads**. The main new ingredients of our generalisation are as follows.

Key words and phrases: operational semantics; category theory.

An extended abstract of this paper appeared in FSCD '20 [41]. The present version incorporates several improvements and additions, listed at the end of §1.

LOGICAL METHODS IN COMPUTER SCIENCE DOI:10.2168/LMCS-???

© A. Hirschowitz, T. Hirschowitz, and A. Lafont
Creative Commons

- We now have two kinds of terms, called **placetakers** and **states**: variables are placeholders for our placetakers, while transitions relate states. Typically, in call-by-value, small-step λ-calculus, placetakers are values, while states are general terms.
- We also have a set of types for placetakers, and a possibly different set of types for transitions and states: transitions of a given type relate states of this type. Typically, in the simply-typed, call-by-value λ-calculus, both sets of types coincide and are given by simple types; while in pure $\bar{\lambda}\mu$-calculus, we have two placetaker types, one for programs and one for stacks, and three transition types, respectively for programs, stacks, and commands.
- We in fact have two possibly different kinds of states for each transition type, **source** states and **target** states, so that a transition of a given type now relates a source state to a target state of this type. Typically, in untyped, call-by-value, big-step λ-calculus, source states (of the unique transition type) are general terms, while target states are values.
- Our variables form a (variable!) family of sets indexed by the placetaker types. To such a variables family X, a transition monad assigns
  - an object $T(X)$ ('of placetakers with free variables in X'), which is again a family of sets indexed by the placetaker types, and
  - two state objects $S_1(T(X))$ and $S_2(T(X))$ ('of source (resp. target) states with free variables in X'), which are families of sets indexed by transition types; here S_1 and S_2 are two functors producing state objects out of placetaker objects.

Roughly speaking (see §3), a transition monad consists of three components:

- a **placetaker** monad T,
- two **state** functors S_1, S_2,
- a **transition structure** consisting of a T-module R and two T-module morphisms $src\colon R \to S_1T$ and $tgt\colon R \to S_2T$,

where T-modules [42] are objects equipped with substitution by elements of T. We view the transition structure (R, src, tgt) as an object of the slice category $T\text{-}\mathbf{Mod}_f/S_1T \times S_2T$ of (finitary) T-modules over $S_1T \times S_2T$.

Reduction monads [9] correspond to the untyped case with $S_1 = S_2 = \mathrm{Id}_{\mathbf{Set}}$. There, reduction monads are identified with suitable **relative monads** [10], and we provide a similar identification for transition monads (see Proposition 3.6).

We present our series of examples of transition monads in §4: $\overline{\lambda}\mu$-calculus, simply-typed λ-calculus (in its call-by-value, big-step variant), π-calculus (as an unlabelled transition system), positive GSOS systems, and differential λ-calculus.

After defining transition monads, we embark on a second part of the paper, devoted to offering the operational semanticist some hopefully convenient tools for defining programming languages (as transition monads).

Specifically, we propose an approach to the specification of transition monads via signatures, which follows the spirit of Initial Semantics [36]. This approach is thus categorical in nature, hence we have to upgrade our sets of transition monads into categories, say $\mathbf{TransMnd}(\mathbb{P}, \mathbb{S})$, one for each pair $(\mathbb{P}, \mathbb{S})$ of sets of placetaker types and transition types (see Definition §5.13). For these categories, we propose what we call a **register**: for a category $\mathbf{C}$, a register consists of

(1) a set **Sig** of **signatures**,

(2) a **semantics** map $[\![-]\!]$ assigning to each signature S in **Sig** a category S-alg of **algebras**, or **models**, together with a **forgetful**[1] functor $U\colon S\text{-alg} \to \mathbf{C}$, and
(3) a **validity** proof of the fact that for each S, the category S-alg has an initial object.

Remark 1.1. This definition is written in type-theoretic style, in the sense that the validity proof is treated as a proper mathematical object. The reader working in a standard, set-theoretical logical setting should of course understand this as an additional condition that registers must satisfy.

A register for $\mathbf{C}$ yields a "decoding" map *spec* sending S to $spec(S) := U(0)$, where 0 here denotes the initial object in S-alg. The efficiency of a register for a category lies in the fact that it allows the expert to easily formalise the informal specification she has in mind for the relevant object. We illustrate in §11 the expressiveness of our register $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$ for transition monads by designing signatures for our examples of §4. For Positive GSOS systems, we even go further by defining a specific register, in which each individual system is more easily specified.

Our register for transition monads is built out from three intermediate registers, corresponding to the three components listed above:

- a register $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$ for monads in the category $\mathbf{Set}^{\mathbb{P}}$,
- a register $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ for functors between $\mathbf{Set}^{\mathbb{P}}$ and $\mathbf{Set}^{\mathbb{S}}$, and
- a register $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ for transition structures over (T, S_1, S_2).

And as could be expected, a signature for a transition monad in our register $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$ is a record consisting of

- a signature of $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$ specifying the placetaker monad T,
- two signatures of $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ specifying the state functors S_1 and S_2, and
- a signature of $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ specifying the transition structure.

Here, a crucial feature is that the involved register in the last field of this record depends on the objects specified by previous fields. This contrasts with the approach taken in [9], where the corresponding field cannot be expressed in terms of the specified placetaker monad, and must instead be parametric in the models of previous fields. Our choice allows more signatures, and we take advantage of this subtle fact in our treatment (see §11.5) of Ehrhard and Regnier's differential λ-calculus [25]. Let us mention that the counterpart for this advantage is that the recursion principle induced by our initial semantics is significantly weaker than could be expected (see Remark 5.14 and §12).

Our signatures for functors and monads incorporate equations similar to those of equational systems: a signature without equations specifies a kind of free object, and adding equations specifies a kind of quotient of this free object, obtained by somehow forcing the added equations to hold. Such a quotienting procedure has already been achieved in a fairly general (and elaborate) way by Fiore and Hur [28]. Under mild additional hypotheses on the ambient category and on equations, we obtain a compact "free+quotient" description of our initial models (Theorems 7.22 and 8.21).

This description roughly goes as follows: we consider a signature augmented with one formal operation for each equation, which yields a new set of terms, say **augmented** terms. By interpreting the new operations as prescribed by each side of the equations, we obtain two translations, say L and R, from augmented terms to plain terms. The initial model

[1]Here "algebra" and "forgetful" have no technical meaning and are chosen by analogy.

is then obtained by identifying any two plain terms of the form $L(a)$ and $R(a)$, for some augmented term a.

This description is a crucial ingredient of our treatment of the differential λ-calculus: as already mentioned, the type of the third field of its signature depends in particular on the monad specified by the first field, and the construction of the signature for this third field relies on our explicit description for that monad (see the proof of Lemma 11.2).

Along the elaboration of our registers, we strive to offer the operational semanticist intuitive notation for defining signatures. They may thus specify programming languages as transition monads by the following procedure:

- Fix sets $\mathbb{P}$ and $\mathbb{S}$ of placetaker and transition types.
- Pick a signature of $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$ for specifying the placetaker monad T, by operations and equations (e.g., using Notation 7.16).
- Choose two signatures of $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ for specifying the state functors S_1 and S_2, again by operations and equations (e.g., using Notation 8.16).
- Choose a signature of $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ for specifying the transition structure, e.g., by giving some rules using the notation of § 6.4.

Plan. In §2, we present our notations and give some categorical preliminaries. In §3 we define transition monads and in §4, we present our selected examples (in the traditional way). Then, in §5, we introduce registers and define our register for transition monads, deferring the precise definition of its components to the next sections. We then introduce our registers for transition structures (§6), monads (§7), and functors (§8), stating the announced explicit descriptions of initial algebras along the way. It remains to prove the latter (and the validity of all registers). We prepare the grounds in §9 by introducing three intermediate registers, and tackle the remaining proofs in §10. In the latter section, we further establish the announced explicit descriptions of initial models. In §11, we then revisit all examples from §4, specifying them through signatures of our registers $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$ for transition monads. Finally, we conclude in §12 by summing up our contributions, assessing the scope of our registers $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$, and giving some perspectives.

Remark 1.2. The reader only interested in using our framework may safely skip §9 and §10.

Related work. As mentioned above, our work refines a recent work [9], allowing to cover many new applications with a very similar approach. This approach differs from the bialgebraic one introduced long ago by Plotkin and Turi [60]: the positive difference is that it covers higher-order languages like the λ-calculus, while a negative difference is that it does not recover congruence of bisimilarity.

Regarding syntax with variable binding, we model it using monads, following a standard approach going back to Bellegarde and Hook [17] (but see also [12, 43, 8]). Because the monad-based approach is essentially equivalent to the presheaf-based one [29, 27], we anticipate that our whole framework could be straightforwardly adapted to presheaf models. We are more cautious about nominal sets [33], mainly because we would need a better understanding of the status of substitution in the latter approach (see, e.g., Power [55]).

Furthermore, our main register for monads (§7) is a simply-typed refinement of [8], close in spirit to [6]. Its validity proof relies on another, new register (§9), which combines two standard registers from the literature, respectively based on equational systems [28]

and pointed strong endofunctors [29]. We could in fact define a register based on equational systems, and use it directly for specifying monads. Moreover, our explicit description of initial models applies in particular to equational systems (Theorem 10.14).

Concerning our registers for slice categories, we suspect that they could be understood in terms of polynomial functors [48], which were recently used in a similar context by Arkor and Fiore [14].

We are aware of a few notions of signatures for languages with variable binding equipped with some notion of evaluation (or transition), e.g., Hamana [37, 38], T. Hirschowitz [44], and Ahrens [6]. They essentially model rewriting, which implies that transitions are closed under arbitrary contexts. Such approaches cannot cover languages like the π-calculus, in which transitions may not occur under inputs or outputs.

As is well-known, evaluation is an oriented variant of equality, and transition proofs are an oriented form of identity proofs. For this reason, notions of signatures for dependently-typed languages like type theory may provide an alternative approach to the specification of operational semantics systems. Examples of such notions of signatures include Fiore's simple dependently-sorted signatures [27], Altenkirch et al.'s categorical semantics of inductive-inductive definitions [11], and Garner's combinatorial approach [34].

Finally, let us mention that a preliminary version of the present work appears in the third author's PhD thesis [50, Chapter 6].

Differences with conference version. Here is a list of significant differences w.r.t. the conference version, beyond more detailed proofs.

- Most importantly, provide an explicit description of the initial algebras of signatures involving equations in §10.
- We also correct a few errors, notably:
  - We had omitted the congruence rules for reduction in $\bar{\lambda}\mu$-calculus and differential λ-calculus.
  - We had omitted the syntactic equation $D(De \cdot f) \cdot g \equiv D(De \cdot g) \cdot f$ from our definition of differential λ-calculus.
  - Again about differential λ-calculus, we had also erroneously claimed that the method we used for defining unary multiterm substitution applies to partial derivation. We now rely on the explicit descriptions of §10 for both operations.
- Finally, we include a few minor improvements, e.g.:
  - We design an abstract version of the original register for slice module categories. In passing, the new version is slightly more expressive.
  - We design a new register combining the features of equational systems and pointed strong endofunctors into **monoidal equational systems** (§9.3).
  - We provide an alternative way in which to organise the π-calculus as a transition monad.

2. Notations and categorical preliminaries

In this section, we fix some notation, and recall a few categorical notions. We advise the reader to skip it, except perhaps §2.1, and get back to it when needed. In §2.1, we fix some basic notation. In §2.2, we recall some well-known results about locally finitely presentable categories. Then, in §2.3, we introduce a notion of *convenient* monoidal category, which is just a monoidal category with additional hypotheses. This is important for us because we

will use finitary monads a lot, and these are monoids for the composition monoidal structure on the category of finitary endofunctors. We show in particular that this composition monoidal structure is convenient. We go on in §2.4 by recalling the standard notion of module over a monoid in a monoidal category, and introduce a notion of parametric module, roughly a module definable for all monoids. We then give a slightly more general definition of modules in the particular case of modules over monads (i.e., when the base category is a functor category). In §2.5, after briefly recalling Beck's monadicity theorem, we state one of its useful (folklore) consequences, a "cancellation" property for monadic functors. In §2.6, we recall the well-known correspondence between finitary monadic functors and finitary monads. Finally, in §2.7, we recall the important fact that equalisers of (finitary) monadic functors are computed as in **CAT**.

2.1. **Basic notation.** In the following, **Set** denotes the category of sets, and **CAT** denotes the (very large) category of locally small categories. We often implicitly view a natural number as an interval $\{1, \dots, n\}$ in $\mathbb{N}$, so that, e.g., $2 \in 3$, $2 \subseteq 3$, and so on. Furthermore, in any category with binary products, we denote by $\langle f, g \rangle \colon C \to A \times B$ the pairing induced by any morphisms $f \colon C \to A$ and $g \colon C \to B$. Similarly, when it makes sense, the copairing of $f \colon A \to C$ and $g \colon B \to C$ is denoted by $[f, g] \colon A + B \to C$. Initial objects are denoted by 0, or by $\emptyset$. Given an endofunctor F, the category of algebras $Fx \to x$ is denoted by F-alg. When F is a monad, the notation F-alg rather refers to its category of algebras in the sense of monads, that is, morphisms $Fx \to x$ satisfying the three standard axioms. Finally, given an object c of a category C, we denote the slice (resp. coslice) category over (resp. under) c by C/c (resp. c/C).

2.2. **Locally finitely presentable categories and finitary functors.** We heavily rely on the theory of locally finitely presentable categories [3]. Very briefly, recall that filtered categories are a categorical generalisation of directed posets. A category is **filtered** when

- it is not empty,
- for any two objects C and D, there is an object E and arrows $C \to E$ and $D \to E$, and furthermore,
- any two parallel arrows $C \rightrightarrows D$ are coequalised by some morphism $D \to E$.

A **filtered colimit** is a colimit of some functor from some small filtered category.

Definition 2.1. An object of a category is **finitely presentable** iff its covariant hom-functor preserves filtered colimits.

Definition 2.2. [3, Definition 1.9] A locally small category is **locally finitely presentable** iff it is cocomplete and every object is a filtered colimit of objects from a fixed set of finitely presentable generators.

Example 2.3 ([3, Example 1.12])**.** Any presheaf category is locally finitely presentable.

Proposition 2.4 ([3, Corollary 1.28])**.** *Any locally finitely presentable category is complete.*

In this context, functors that preserve filtered colimits are important. They are called **finitary**.

Definition 2.5. Let $[\mathbf{C}, \mathbf{D}]_f$ denote the category of finitary functors $\mathbf{C} \to \mathbf{D}$ for any categories $\mathbf{C}$ and $\mathbf{D}$.

Proposition 2.6. *If* $\mathbf{C}$ *and* $\mathbf{D}$ *are locally finitely presentable, then so is* $[\mathbf{C}, \mathbf{D}]_f$.

Proof. See [47, Section 4] for the more general enriched case. □

2.3. **Convenient monoidal categories.** We sometimes work in a monoidal category satisfying some additional properties. We call such monoidal categories convenient.

Definition 2.7. A monoidal category is **convenient** when

- it is locally finitely presentable;
- the tensor preserves filtered colimits on the right and all colimits on the left.

Proposition 2.8. *Any category* $[\mathbf{C}, \mathbf{C}]_f$ *of finitary endofunctors on a locally finitely presentable category* $\mathbf{C}$ *is convenient for the composition monoidal structure.*

Proof. By Proposition 2.6, any category of finitary endofunctors on a locally finitely presentable category is locally finitely presentable. Furthermore, since colimits are computed pointwise in functor categories whenever the codomain category is cocomplete [52, §V.4], we have $(\mathrm{colim}_i G_i) \circ F \cong \mathrm{colim}_i (G_i \circ F)$ for any diagram G and object F, thus the composition tensor product preserves all colimits on the left. Finally, by definition of finitariness, the considered functors preserve filtered colimits, hence for any such diagram G and object F we have $F \circ \mathrm{colim}_i G_i \cong \mathrm{colim}_i (F \circ G_i)$ as desired. □

Example 2.9. In particular, all categories of the form $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$ that we will consider below (where $\mathbb{P}$ is a set) are convenient for the composition monoidal structure.

2.4. **(Parametric) modules over monoids and monads.** Let us fix a monoidal category $\mathbf{C}$. We first recall the standard notion of (right) module over a monoid, and introduce the notion of parametric module, inspired by [42].

Definition 2.10. Given a monoid $X \in \mathbf{C}$, a (right) X**-module** is an object M equipped with a morphism $M \otimes X \to M$ satisfying natural coherence conditions (the axioms for the dual case of left modules are given in [58, §3.2]). We denote by X-**Mod** the category of X-modules, with action-preserving morphisms between them.

Remark 2.11. The coherence conditions amount to equipping M with algebra structure for the monad $- \otimes X$ induced by X.

A parametric module will assign a module to any monoid. In order to formally define this, let us introduce the following category, which collects all categories of the form X-**Mod**.

Definition 2.12. Let $\mathbf{Mod}(\mathbf{C})$ denote the category

- whose objects are pairs (X, M) where X is a monoid and M is an X-module, and
- whose morphisms $(X, M) \to (Y, N)$ are pairs (f, g) of a monoid morphism $f : X \to Y$ and a morphism $g : M \to N$ in $\mathbf{C}$ such that the following diagram commutes.

$$\begin{array}{ccc} M \otimes X & \xrightarrow{g \otimes f} & N \otimes Y \\ \downarrow & & \downarrow \\ X & \xrightarrow[g]{} & Y \end{array}$$

Let $\mathbf{U}^{\mathbf{Mod}} : \mathbf{Mod}(\mathbf{C}) \to \mathbf{Mon}(\mathbf{C})$ denote the forgetful functor.

Definition 2.13. A **parametric module** over $\mathbf{C}$ is a section of the forgetful functor $\mathbf{U}^{\mathbf{Mod}}\colon \mathbf{Mod}(\mathbf{C}) \to \mathbf{Mon}(\mathbf{C})$, i.e., a functor $S\colon \mathbf{Mon}(\mathbf{C}) \to \mathbf{Mod}(\mathbf{C})$ such that $\mathbf{U}^{\mathbf{Mod}} \circ S = \mathrm{id}_{\mathbf{Mon}(\mathbf{C})}$.

In other words, a parametric module functorially assigns to each monoid a module over it.

Example 2.14. If $\mathbf{C}$ has products, then we can define the parametric module mapping a monoid X to the X-module $X \times X$. It will become clear in §7 how this parametric module can be viewed as the arity of a binary operation, with $\mathbf{C} = [\mathbf{Set}, \mathbf{Set}]_f$.

Example 2.15. Any endofunctor T on $\mathbf{Mon}(\mathbf{C})$ equipped with a natural transformation $\eta : \mathrm{Id} \to T$ induces a parametric module T^{monmod} mapping any monoid M to $T(M)$, with action given by

$$T(M) \otimes M \xrightarrow{T(M)\otimes\eta_M} T(M) \otimes T(M) \to T(M),$$

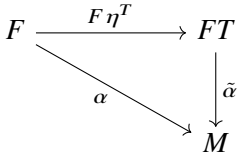
where the second morphism is the multiplication of $T(M) \in \mathbf{Mon}(\mathbf{C})$. This construction applies in particular for any monad on $\mathbf{Mon}(\mathbf{C})$.

The previous definitions of (parametric) modules specialise to the case where $\mathbf{C} = [\mathbf{E}, \mathbf{E}]_f$ (for the composition monoidal structure). Then, monoids are finitary monads. However, because in this case $\mathbf{C}$ is an endofunctor category, there is a slightly more general, "relative", or "heterogeneous" notion of module [42] which will be important for us. Let us recall it now.

Definition 2.16. Given a monad T on $\mathbf{C}$ and a category $\mathbf{D}$, a **D-valued T-module** is a finitary functor $M : \mathbf{C} \to \mathbf{D}$ equipped with a right T-action $M \circ T \to M$ satisfying coherence conditions analogous to those of Definition 2.10. A morphism of T-modules is similarly a natural transformation commuting with action. We denote by $T\text{-}\mathbf{Mod}_f(\mathbf{D})$ the category of T-modules and morphisms between them.

Remark 2.17. $\mathbf{D}$-valued T-modules are algebras for the monad $- \circ T$ on $[\mathbf{C}, \mathbf{D}]_f$. When $\mathbf{D} = \mathbf{C}$, these are the same as T-modules in $[\mathbf{C}, \mathbf{C}]_f$ in the sense of Definition 2.10.

Notation 2.18. *Given a monad T on $\mathbf{C}$, and functor $F : \mathbf{C} \to \mathbf{D}$, we sometimes implicitly equip the functor FT with its canonical T-module structure. This module is free on F, in the sense that for any T-module M and natural transformation $\alpha\colon F \to M$, there is a unique T-module morphism $\tilde{\alpha}\colon FT \to M$ making the following diagram commute.*



Of course, $\tilde{\alpha}$ is merely the composite $FT \xrightarrow{\alpha_T} MT \to M$.

Let us briefly show how to exploit the variability of $\mathbf{D}$.

Definition 2.19. For any p in a set $\mathbb{P}$, and any monad T on $\mathbf{Set}^{\mathbb{P}}$, the functor $T_p\colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}$ mapping any $X \in \mathbf{Set}^{\mathbb{P}}$ to $T(X)(p)$ is a T-module.

Here is another example construction of T-module, which is useful for specifying syntax with variable binding.

Definition 2.20. For any sequence $p_1, \ldots, p_n$ in a set $\mathbb{P}$, for any monad T on $\mathbf{Set}^{\mathbb{P}}$ and $\mathbf{D}$-valued T-module M, we denote by $M^{(p_1,\ldots,p_n)}$ the $\mathbf{D}$-valued T-module defined by

$$M^{(p_1,\ldots,p_n)}(X) = M(X + \mathbf{y}_{p_1} + \ldots + \mathbf{y}_{p_n}),$$

where $\mathbf{y} : \mathbb{P} \to \mathbf{Set}^{\mathbb{P}}$ is the embedding defined by $\mathbf{y}_p(q) = 1$ if $p = q$ and $\emptyset$ otherwise. If $\mathbb{P}$ is a singleton, we abbreviate this to $M^{(n)}$.

Following up on Remark 2.17, besides the notion of (homogeneous) parametric module in the sense of Definition 2.13, there is for any $\mathbf{D}$ a notion of $\mathbf{D}$-valued parametric module, spelled out in §7.1.

2.5. **Creation of (co)limits and monadic functors.** In this section, we recall Beck's monadicity theorem [52, Theorem VI.7.1], and state a "cancellation" property for monadic functors. First, we need to recall creation of (co)limits, monadic functors, and absolute (co)limits. We will see in §10 that the main technical notion for our explicit descriptions of initial algebras is creation of (co)limits, so let us briefly recall the basics.

Definition 2.21. Given a small category $\mathbf{E}$, a functor $F\colon \mathbf{C} \to \mathbf{D}$ **creates** (co)limits of shape $\mathbf{E}$ if for any functor $J\colon \mathbf{E} \to \mathbf{C}$, if FJ has a (co)limit $D \in \mathbf{D}$, then the (co)limiting (co)cone uniquely lifts to $\mathbf{C}$, and the lifting is again (co)limiting.

A typical example is:

Proposition 2.22. [52, Exercise VI.2.2] *For any monad T on a category $\mathbf{C}$, the forgetful functor $U\colon T\text{-alg} \to \mathbf{C}$ creates limits.*

We also have the following well-known result:

Proposition 2.23. *For any monad T on a category $\mathbf{C}$, the forgetful functor $U\colon T\text{-alg} \to \mathbf{C}$ creates colimits of a given shape whenever T preserves them. More concretely, if T preserves all colimits of functors with some domain $\mathbf{D}$, then U creates them.*

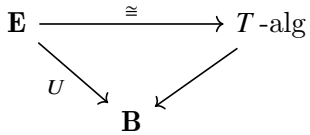
Proof. This is a straightforward consequence of [22, Proposition 4.3.2]. □

Corollary 2.24. *For any monad T on a cocomplete category $\mathbf{C}$, and for any category $\mathbf{D}$, the following are equivalent*

(i) *the forgetful functor $U\colon T\text{-alg} \to \mathbf{C}$ creates all colimits of shape $\mathbf{D}$, and*
(ii) *the monad T preserves all colimits of shape $\mathbf{D}$.*

Proof. Proposition 2.23 proves *(ii) ⇒ (i)*. For the converse, if U creates all colimits of shape $\mathbf{D}$, then by cocompleteness of $\mathbf{C}$, it also preserves them. Letting L denote the left adjoint to U, we then get that $T = UL$ preserves all such colimits by composition, as left adjoints are cocontinuous. □

Definition 2.25. A functor $U\colon \mathbf{E} \to \mathbf{B}$ is **monadic** if $\mathbf{E}$ is isomorphic to a category of algebras T-alg for some monad T on $\mathbf{B}$, and furthermore, the following diagram commutes.



Remark 2.26. This notion of monadicity is quite strict. Some authors prefer a relaxed version where isomorphism is replaced with equivalence.

Definition 2.27. A (co)limit is **absolute** if it is preserved by all functors.

Example 2.28. A split coequaliser is a coequaliser

$$A \underset{g}{\overset{f}{\rightrightarrows}} B \xrightarrow{e} C,$$

such that

- e has a section $s : C \to B$,
- f has a section $t : B \to A$ with
- $g \circ t = s \circ e$.

Such a coequaliser is absolute [52, Corollary VI.6].

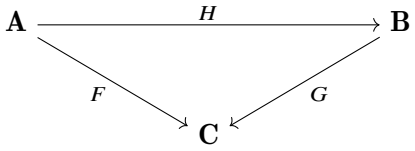
On the other hand, the initial object $\emptyset$ in **Set**, viewed as the colimit of the unique functor from the empty category, is not absolute, since it is not preserved by the constant terminal endofunctor on **Set**.

Theorem 2.29 (Beck's monadicity theorem [52, Theorem VI.7])**.**
A functor $U\colon \mathbf{E} \to \mathbf{B}$ is monadic iff

- *it has a left adjoint, and*
- *it creates coequalisers for those parallel pairs $f, g\colon E_1 \rightrightarrows E_2$ for which $(U(f), U(g))$ has an absolute coequaliser in $\mathbf{B}$.*

We will use the following consequence.

Proposition 2.30. *Given a commuting triangle of functors*



between cocomplete categories, if F and G are monadic, then so is H.

If furthermore F and G are finitary, then so is H.

Proof. For monadicity, Borceux [22, Corollary 4.5.7] gives a proof in the weakly monadic case (see Remark 2.26). This is a straightforward adaptation. Finitariness follows from the next lemma. □

Lemma 2.31. *Given a commuting triangle of functors as in Proposition 2.30, if F is finitary and G creates filtered colimits, then H is finitary.*

Proof. Given a colimiting cocone $c\colon J \to A$ for a filtered diagram $J\colon \mathbf{D} \to \mathbf{A}$, this cocone is preserved by F and created by G. So $F(c)$ is colimiting, and has a unique antecedent by G, which is again colimiting. But $H(c)$ is an antecedent, hence has to be *the* antecedent, and so is colimiting as desired. □

2.6. **Monads vs. monadic functors.** In this section, we recall the equivalence between (finitary) monadic functors and (finitary) monads.

Let us fix a locally finitely presentable category **C**.

Definition 2.32. Let $\mathbf{Mnd}_f(\mathbf{C})$ denote the category of finitary monads on **C**. When **C** is clear from context, we sometimes abbreviate this to just $\mathbf{Mnd}_f$.

Definition 2.33. Let $\mathbf{Monadic}_f/\mathbf{C}$ denote the full subcategory of $\mathbf{CAT}/\mathbf{C}$ spanning (strictly) monadic functors that are finitary, or equivalently (by Corollary 2.24), whose underlying monad is finitary.

Let us readily make the following observation.

Lemma 2.34. *The functor underlying any morphism in* $\mathbf{Monadic}_f/\mathbf{C}$ *is itself monadic.*

Proof. By Proposition 2.30. □

In fact, a lot of our understanding of $\mathbf{Monadic}_f/\mathbf{C}$ will follow from the following equivalence.

Proposition 2.35. *The functor*

$$(-)\text{-alg}\colon \mathbf{Mnd}(\mathbf{C})^{op} \to \mathbf{Monadic}/\mathbf{C}$$

mapping any monad T *to the forgetful functor* $T\text{-alg} \to \mathbf{C}$ *is an equivalence.*

Proof. The functor is essentially surjective by definition of monadic functors. It is also full and faithful by [15, Proposition 5.3]. □

Corollary 2.36. *The functor*

$$(-)\text{-alg}\colon \mathbf{Mnd}_f(\mathbf{C})^{op} \to \mathbf{Monadic}_f/\mathbf{C}$$

mapping any finitary monad T *to the forgetful functor* $T\text{-alg} \to \mathbf{C}$ *is an equivalence.*

Proof. The equivalence of Proposition 2.35 restricts to the desired one by Corollary 2.24. □

Notation 2.37. *We denote by* $U \mapsto \check{U}$ *the (or, rather, some chosen) inverse equivalence.*

2.7. **Limits of finitary monadic functors.** In this section, we recall well-known results about limits of (finitary) monadic functors, namely that they are computed as in $\mathbf{CAT}$, or more precisely that the forgetful functor $\mathbf{Monadic}_f/\mathbf{C} \to \mathbf{CAT}/\mathbf{C}$ creates limits. For this, we need the following two lemmas.

Lemma 2.38. *Finitary monads over any locally finitely presentable category form a locally finitely presentable category.*

Proof sketch (see [49]*).* Let $\mathbf{C}$ be locally presentable. Then $\mathbf{C}_f$ is small, so $[\mathbf{C}_f, \mathbf{C}]$ is again locally presentable. Now, finitary monads on $\mathbf{C}$ are equivalently monoids for the composition tensor product in $[\mathbf{C}_f, \mathbf{C}]$, hence also algebras for a finitary monad on a locally presentable category, which allows us to conclude by the following lemma. □

Lemma 2.39. *The category of algebras for a finitary monad on any locally finitely presentable category is again locally finitely presentable.*

Proof. This is [3, Remark 2.78] with $\lambda = \omega$. □

Now a useful tool to prove that our categories of models are monadic is the following result and its corollary, coming up just next.

Proposition 2.40. *The forgetful functor* $\mathbf{Monadic}_f/\mathbf{C} \to \mathbf{CAT}/\mathbf{C}$ *creates limits.*

For the proof, we first need two definitions and a lemma.

Definition 2.41. A functor $F\colon \mathbf{C} \to \mathbf{D}$ is **amnestic** iff, for any isomorphism $i\colon C \to C'$ in $\mathbf{C}$ such that $F(C) = F(C')$ and $F(i) = \mathrm{id}$, we have $C = C'$ and $i = \mathrm{id}$.

Definition 2.42. A functor $F\colon \mathbf{C} \to \mathbf{D}$ is an **iso-fibration** iff, for any isomorphism of the form $i\colon D \to F(C)$, there exists an isomorphism $j\colon C' \to C$ such that $F(j) = i$.

Lemma 2.43. *Any continuous, amnestic iso-fibration from a complete category creates limits.*

Proof. Consider any continuous, amnestic iso-fibration $\mathbf{F}\colon \mathbf{C} \to \mathbf{D}$ with $\mathbf{C}$ complete, and any functor $J\colon \mathbf{X} \to \mathbf{C}$ such that FJ has a limiting cone, say $\delta\colon D \to FJ$. Then, because $\mathbf{C}$ is complete, J also has a limiting cone, say $\gamma\colon C \to J$. Because F is continuous, $F(\gamma)\colon F(C) \to FJ$ is again limiting, hence we get an isomorphism $i\colon D \to F(C)$ of cones over FJ. Because F is an iso-fibration, we then lift i to an isomorphism $j\colon C' \to C$ in $\mathbf{C}$ such that $F(j) = i$. The cone $\gamma j\colon C' \to J$ is thus limiting, and an antecedent of δ. It thus remains to show that it is the only antecedent ofδ. Let thus $\gamma''\colon C'' \to J$ be any antecedent of δ. Because γj is limiting, there exists a unique cone morphism $m\colon C'' \to C'$. But now $F(m)$ is a cone endomorphism $\delta \to \delta$, hence $F(m) = \mathrm{id}$. Because F is amnestic, we then get $C'' = C'$ and $m = id$, thus proving that $\gamma'' = \gamma j$ as desired. □

Proof of Proposition 2.40. By Lemma 2.38 and Proposition 2.35, $\mathbf{Monadic}_f/\mathbf{C}$ is complete, and equivalent to the opposite category of finitary monads on $\mathbf{C}$.

Moreover, the forgetful functor $\mathbf{Monadic}_f/\mathbf{C} \to \mathbf{CAT}/\mathbf{C}$ is continuous. Indeed, it is equivalent (in $\mathbf{CAT}^{\to}$) to the composite

$$\mathbf{Mnd}_f(\mathbf{C})^{op} \hookrightarrow \mathbf{Mnd}(\mathbf{C})^{op} \to \mathbf{CAT}/\mathbf{C},$$

whose first component is continuous by [18, Proposition 5.6], while the second is by [46, Proposition 26.3].

Finally, the forgetful functor $\mathbf{Monadic}_f/\mathbf{C} \to \mathbf{CAT}/\mathbf{C}$ is an amnestic iso-fibration:

- it is amnestic as a subcategory embedding, and
- an iso-fibration because the subcategory in question is replete (otherwise said, monadic functors are closed under isomorphisms).

The result thus follows by Lemma 2.43. □

Corollary 2.44. *The forgetful functor* $\mathbf{Monadic}_f/\mathbf{C} \to \mathbf{CAT}$ *creates equalisers. More precisely, given monadic functors* $F_1\colon \mathbf{D}_1 \to \mathbf{C}$ *and* $F_2\colon \mathbf{D}_2 \to \mathbf{C}$ *and functors* $G_1, G_2\colon \mathbf{D}_1 \to \mathbf{D}_2$ *such that* $F_2 \circ G_i = F_1$*, if* $\mathbf{A} \to \mathbf{D}_1$ *is the equaliser of* G_1 *and* G_2 *in* $\mathbf{CAT}$*, then the composite* $\mathbf{A} \to \mathbf{D}_1 \to \mathbf{C}$ *is finitary monadic and underlies the equaliser of* G_1 *and* G_2 *in* $\mathbf{Monadic}_f/\mathbf{C}$*.*

Proof. This follows straightforwardly from Proposition 2.40 and the fact that the forgetful functor from any slice category to the base category creates equalisers (see the proof of [21, Proposition 2.16.3]). □

3. Transition monads

In this section, we introduce the main new mathematical notion of the paper: transition monads. In §3.1 we give an informal description and in §3.2, we give our formal definition. In §3.3, we provide an equivalent definition based on the notion of relative monad. Finally, in §3.4, we sketch a proof-irrelevant variant of our notion.

3.1. **Overview of transition monads.**

Placetakers and states

In standard λ-calculus, we have terms, variables are placeholders for terms, and transitions relate a source term to a target term. In a general transition monad we still have variables and transitions, but placetakers for variables and endpoints of transitions can be of a different nature, which we phrase as follows: variables are placeholders for **placetakers**, while transitions relate a **source state** with a **target state**.

The categories for placetakers and for states

In standard λ-calculi, we have a set $\mathbb{T}$ of types for terms (and variables); for instance in the untyped version, $\mathbb{T}$ is a singleton. Accordingly, terms form a **monad** on the category $\mathbf{Set}^{\mathbb{T}}$. In a general transition monad we have a set $\mathbb{P}$ of placetaker types, and placetakers form a **monad** on the category $\mathbf{Set}^{\mathbb{P}}$; similarly, we have a set $\mathbb{S}$ of transition types, and the category of states is $\mathbf{Set}^{\mathbb{S}}$. For example, for simply-typed λ-calculus, $\mathbb{P} = \mathbb{S}$ is the set of simple types.

The object of variables

In our view of the untyped λ-calculus, there is a (variable!) set of variables and everything is parametric in this 'variables set'. Similarly, in a general transition monad R, there is a 'variables object' V in $\mathbf{Set}^{\mathbb{P}}$ and everything is functorial in this variables object. In particular, we have a placetaker object $T_R(V)$ in $\mathbf{Set}^{\mathbb{P}}$ and a source (resp. target) state object in $\mathbf{Set}^{\mathbb{S}}$, both depending upon the variables object.

The state functors S_1 and S_2

While in the λ-calculus, states are the same as placetakers, in a general transition monad, they may differ, and more precisely both state objects are derived from the placetaker object by applying the **state functors** $S_1, S_2 \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$.

The transition structure

In standard λ-calculi, there is a (typed!) set of transitions, which yields a graph on the set of terms. That is to say, if V is the variables object, and $LC(V)$ the placetaker object, there is a transition object $Trans(V)$ equipped with two maps $src_V, tgt_V : Trans(V) \to LC(V)$. Note that we consider 'proof-relevant' transitions here, in the sense that two different transitions may have the same source and target (see 3.4 for the irrelevant variant).

In a general transition monad R, we still have a transition object $Trans_R(V)$, which now lives in $\mathbf{Set}^{\mathbb{S}}$, together with state objects $S_1(T_R(V))$ and $S_2(T_R(V))$, so that the pairing $\langle src_V, trg_V \rangle$ forms a morphism $Trans_R(V) \to S_1(T_R(V)) \times S_2(T_R(V))$.

3.2. **The definition of transition monad.** Here is our formal definition:

Definition 3.1. Given two sets $\mathbb{P}$ and $\mathbb{S}$, a **transition monad** over $(\mathbb{P}, \mathbb{S})$ consists of

- a finitary monad T on $\mathbf{Set}^{\mathbb{P}}$, called the **placetaker** monad,
- two finitary functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, called **state** functors, and
- a **transition structure** $R \xrightarrow{(src,tgt)} S_1T \times S_2T$ consisting of
  - a finitary T-module $R : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, called the **transition** module,
  - a **source** T-module morphism $src : R \to S_1T$, recalling from Notation 2.18 that S_1T is the free T-module on S_1,
  - a **target** T-module morphism $tgt : R \to S_2T$.

Notation 3.2. *For any sets $\mathbb{P}$ and $\mathbb{S}$, finitary monad T over $\mathbf{Set}^{\mathbb{P}}$, and finitary functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, we let $\mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ denote the class of transition structures over T, S_1, and S_2.*

Furthermore, let $\mathbf{TransMnd}_{\mathbb{P},\mathbb{S}}$ denote the coproduct $\sum_{T,S_1,S_2} \mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$.

In Definition §5.13, we will upgrade these classes into categories with the same names.

3.3. **Transition monads as relative monads.** In our definition of transition monad, we have required the two state modules to take the form S_1T and S_2T, but this is not essential in our development. Moreover, it probably leaves some relevant examples out of reach. We think in particular about the standard presentation of the π-calculus as a labelled transition system. Our original reason for this design choice was purely aesthetic: it ensures that our transition monads are **relative** monads, as were the reduction monads introduced in [9] (in particular the untyped λ-calculus). These were monads relative to the 'discrete graph' functor from sets to graphs, and in our extended context, we have to replace graphs by S-**graphs**. Let us provide more detail.

Let us first recall [10] that, given any functor $J : \mathbf{C} \to \mathbf{D}$, a **monad relative to** J, or J-**relative monad**, consists of

- an object mapping $T : \mathbf{ob}(\mathbf{C}) \to \mathbf{ob}(\mathbf{D})$, together with
- morphisms $\eta_X : J(X) \to T(X)$, and
- for each morphism $f : J(X) \to T(Y)$, an **extension** $f^\star : T(X) \to T(Y)$,

satisfying coherence conditions. Any J-relative monad T has an underlying functor $\mathbf{C} \to \mathbf{D}$, and is said **finitary** when this functor is. Note that a monad is nothing but a J-relative monad, for J the identity endofunctor.

Now we define S-graphs:

Definition 3.3. For any pair $S = (S_1, S_2)$ of functors $\mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, an S-**graph** over an object $V \in \mathbf{Set}^{\mathbb{P}}$ consists of

- an object E (of **edges**) in $\mathbf{Set}^{\mathbb{S}}$, and
- a morphism $\partial : E \to S_1(V) \times S_2(V)$.

Accordingly, an S-**graph** consists of an object $V \in \mathbf{Set}^{\mathbb{P}}$ and an S-graph over V.

We can now say that in a general transition monad, transitions form an S-graph over the placetaker object (the whole thing depending upon the variables object...). Before proceeding, we must introduce the category of S-graphs: a morphism $G \to G'$ consists of a morphism for vertices $f : V_G \to V_{G'}$ together with a morphism for edges $g : E_G \to E_{G'}$ making the following diagram commute.

$$\begin{array}{ccc} E_G & \xrightarrow{\quad g \quad} & E_{G'} \\ {\scriptstyle \partial_G}\downarrow & & \downarrow{\scriptstyle \partial_{G'}} \\ S_1(V_G) \times S_2(V_G) & \xrightarrow[S_1(f)\times S_2(f)]{} & S_1(V_{G'}) \times S_2(V_{G'}) \end{array}$$

Proposition 3.4. *For any pair $S = (S_1, S_2)$ of functors $\mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, S-graphs form a category S-**Gph**.*

The proof is a straightforward verification.

We will consider monads relative to the following functors:

Definition 3.5. For any pair $S = (S_1, S_2)$ of functors $\mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, the **discrete S-graph** functor $J_S : \mathbf{Set}^{\mathbb{P}} \to S\text{-}\mathbf{Gph}$ maps any $V \in \mathbf{Set}^{\mathbb{P}}$ to the S-graph on V with no edges.

Now we are ready to deliver our characterisation of transition monads as relative monads:

Proposition 3.6. *Given finitary functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, transition monads with state functors S_1 and S_2 are exactly monads relative to the discrete S-graph functor for $S = (S_1, S_2)$, such that the induced functor $\mathbf{Set}^{\mathbb{P}} \to S\text{-}\mathbf{Gph}$ is finitary.*

The proof consists merely in unfolding the definitions. Since we do not use this result, we do not give further details.

3.4. **The proof-irrelevant variant.** Although we have chosen a "proof-relevant" notion of transition monad, we can sketch a presentation of a "proof-irrelevant" variant.

Definition 3.7. The subset of $\mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ consisting of transition monads $\langle src, tgt\rangle : R \to S_1T \times S_2T$ such that $\langle src, tgt\rangle$ is a pointwise inclusion is denoted by $\mathbf{ITransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$.

Remark 3.8. We have a natural retraction

$$\mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) \twoheadrightarrow \mathbf{ITransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2),$$

which maps a transition structure $\partial : R \to S_1T \times S_2T$ to the monomorphism $\overline{R} \hookrightarrow S_1T \times S_2T$ obtained from the (strong epi)-mono factorisation; this factorisation exists [3, Proposition 1.61] since the category of finitary **Set**-valued T-modules is a presheaf category [5, Definition 2.71]. In Proposition 6.19, we will upgrade this retraction into a coreflection of categories.

4. Examples of transition monads

In this section, we present very informally the announced examples of transition monads. This presentation should eventually be compared to the one via signatures given in §11.

4.1. **The call-by-value, simply-typed, big-step λ-calculus.** The notion of transition monad accounts for many different variants of the λ-calculus. Let us detail the case of the simply-typed, call-by-value, big-step λ-calculus. Most often, big-step semantics describes evaluation of closed terms. Here we consider a variant describing the evaluation of open terms [53, 51]. In this setting, the main subtlety lies in the fact that variables are only placeholders for values.

We fix some set of base types, ranged over by ι and define successively types, values and terms, typing contexts, well-typed terms, and transitions as follows.

Definition 4.1. The set $\mathbb{P}$ of **types**, ranged over by A, B, is defined inductively by

$$A, B ::= \iota \mid A \to B.$$

Definition 4.2. For any set, say X, of **variables**, we then define **values**, ranged over by v, w, and general **terms**, ranged over by e, f in a mutually inductive way as follows.

$$\begin{array}{rcl} v, w & ::= & x \mid \lambda x.e \\ e, f & ::= & v \mid e\ f \end{array}$$

Here, x ranges over X, and Terms are considered equivalent modulo α-conversion. Furthermore, one may define capture-avoiding substitution, as usual.

Definition 4.3. A **typing context** is a type-indexed family of sets, i.e., an object of $\mathbf{Set}^{\mathbb{P}}$. For any $\Gamma \in \mathbf{Set}^{\mathbb{P}}$, $A \in \mathbb{P}$, and $x \notin \Gamma_A$, we let $\Gamma, x : A$ denote Γ augmented with x over A.

Remark 4.4. The extended context $\Gamma, x : A$ is isomorphic to $\Gamma + \mathbf{y}_A$, where $\mathbf{y}$ denotes the Yoneda embedding $\mathbb{P} \hookrightarrow \mathbf{Set}^{\mathbb{P}}$, viewing $\mathbb{P}$ as a discrete category. Indeed, because $\mathbb{P}$ is discrete, $\mathbf{y}_A(B)$ is empty when $A \neq B$, and $\mathbf{y}_A(A)$ is a singleton. Thus $\Gamma + \mathbf{y}_A$ is Γ, plus one element of type A.

Definition 4.5. Well-typed terms are inductively defined by the following rules.

$$\frac{}{\Gamma \vdash x : A}\ (x \in \Gamma_A) \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.e : A \to B} \qquad \frac{\Gamma \vdash e : A \to B \qquad \Gamma \vdash f : A}{\Gamma \vdash e\ f : B}$$

Definition 4.6. Transitions are inductively defined by the following rules.

$$\frac{}{v \Downarrow v} \qquad \frac{e_1 \Downarrow \lambda x.e_3 \qquad e_2 \Downarrow w \qquad e_3[x \mapsto w] \Downarrow v}{e_1\ e_2 \Downarrow v}$$

This calculus forms a transition monad as follows.

Placetakers and transition types As foreshadowed by the notation, because variables and values are indexed by (simple) types, we take $\mathbb{P} = \mathbb{S}$ to be the set of types.

Placetaker monad The placetaker monad T over $\mathbf{Set}^{\mathbb{P}}$ is given by well-typed values: given any $\Gamma \in \mathbf{Set}^{\mathbb{P}}$, the placetaker object $T(\Gamma) \in \mathbf{Set}^{\mathbb{P}}$ assigns to each type A the set $T(\Gamma)_A$ of values v such that $\Gamma \vdash v : A$ holds.

State functors In big-step semantics, transitions relate terms to values. Hence, we are seeking state functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{P}}$ such that $S_1(T(\Gamma))_A$ is the set of λ-terms of type A with free variables in Γ, and $S_2(T(\Gamma))_A$ is the subset of values therein. For S_2, we should clearly take the identity functor since T consists of all values. For S_1, we first observe that λ-terms can be described as **application binary trees** whose leaves are values

(internal nodes being typed applications). More formally, let $S_1(\Gamma)_A$ denotes the set of terms $\Gamma \vdash_{S_1} b : A$, as inductively defined by the following rules.

$$\frac{x \in \Gamma_A}{\Gamma \vdash_{S_1} x : A} \qquad \frac{\Gamma \vdash_{S_1} b : A \to B \qquad \Gamma \vdash_{S_1} b' : A}{\Gamma \vdash_{S_1} b\ b' : B}\ .$$

If Γ is a typing context, $S_1(T(\Gamma))$ is indeed the set of well-typed general terms with variables in Γ.

Transition module Finally, the transition module maps any $\Gamma \in \mathbf{Set}^{\mathbb{P}}$ to the family, over all $A \in \mathbb{P}$, of transition proofs of some $e \Downarrow v$ with $e \in S_1(T(\Gamma))_A$ and $v \in T(\Gamma)_A$. Such transition proofs are stable under value substitution, so we obtain a transition monad.

4.2. **The $\bar{\lambda}\mu$-calculus.** The $\overline{\lambda}\mu$-calculus, introduced by Herbelin [39], models the computational contents of cut elimination in the sequent calculus. Following Vaux's presentation [61, §2.4.4], its grammar is given by

Commands	Programs	Stacks
$c ::= \langle e \mid \pi \rangle$	$e ::= x \mid \mu\alpha.c \mid \lambda x.e$	$\pi ::= \alpha \mid e \cdot \pi,$

where x and α range over two disjoint sets of variables, called **stack** and **program** variables, respectively. Both constructions μ and λ bind their variable in the body. Reduction is generated by the following two basic transition rules concerning commands:

$$\langle \mu\alpha.c \mid \pi \rangle \to c[\alpha \mapsto \pi] \qquad \langle \lambda x.e \mid e' \cdot \pi \rangle \to \langle e[x \mapsto e'] \mid \pi \rangle.$$

Since reduction may occur "everywhere", it concerns programs and stacks as well.

Let us show how this gives rise to a transition monad.

Placetaker types In the transition rules above, we see that placetakers may be programs or stacks. So, we take two placetaker types: $\mathbb{P} := 2 = \{\mathbf{p}, \mathbf{s}\}$. A variables object is an element of $\mathbf{Set}^{\mathbb{P}}$, that is, a pair of sets: the first one gives the available free program variables, and the second one the available free stack variables.

Placetaker monad The syntax may be viewed as a monad $T : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{P}}$: given a variables object $X = (X_{\mathbf{p}}, X_{\mathbf{s}}) \in \mathbf{Set}^{\mathbb{P}}$, the placetaker object $(T(X)_{\mathbf{p}}, T(X)_{\mathbf{s}}) \in \mathbf{Set}^{\mathbb{P}}$ consists of the sets of program and stack terms with free variables in X, up to bound variable renaming. As usual, monad multiplication is given by capture-avoiding substitution.

The T-modules for programs, stacks and commands We have three important **Set**-valued T-modules: the first two, *Prg* for programs and *Stk* for stacks, are the two components of T. The third one, for commands, is just the cartesian product of the previous two: $Cmd := Prg \times Stk$.

Transition types and state functors As mentioned above, transitions concern programs and stacks as well as commands. Thus, we take three transition types: $\mathbb{S} = 3 = \{\mathbf{c}, \mathbf{p}, \mathbf{s}\}$. Furthermore, commands are pairs of a program and a stack, so that, setting $S_1(A) = S_2(A) = (A_{\mathbf{p}} \times A_{\mathbf{s}}, A_{\mathbf{p}}, A_{\mathbf{s}})$, we get $S_i(T(X)) = (T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}}, T(X)_{\mathbf{p}}, T(X)_{\mathbf{s}})$ for $i = 1, 2$, as desired.

Transition module Finally, transitions with free variables in X form a triple of graphs with vertices respectively in $T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}}$ (the set of commands taking free variables in X), $T(X)_{\mathbf{p}}$, and $T(X)_{\mathbf{s}}$. This family is natural in X and commutes with substitution, hence forms a T-module morphism, which completes our transition monad.

4.3. **The π-calculus.** For an example involving equations on placetakers, let us recall the following standard presentation of (a simple variant of) the π-calculus [57]. The syntax for **processes** is given by

$$P, Q ::= 0 \mid (P|Q) \mid \nu a.P \mid \bar{a}\langle b\rangle.P \mid a(b).P,$$

where a and b range over **channel names**, and b is bound in $\nu b.P$ and $a(b).P$. Processes are identified when related by the smallest context-closed equivalence relation $\equiv$ satisfying

$$0|P \equiv P \qquad P|Q \equiv Q|P \qquad P|(Q|R) \equiv (P|Q)|R \qquad (\nu a.P)|Q \equiv \nu a.(P|Q),$$

where in the last equation a should not occur free in Q. Transitions are then given by the following rules.

$$\frac{}{\bar{a}\langle b\rangle.P|a(c).Q \longrightarrow P|(Q[c \mapsto b])} \qquad \frac{P \longrightarrow Q}{P|R \longrightarrow Q|R} \qquad \frac{P \longrightarrow Q}{\nu a.P \longrightarrow \nu a.Q}$$

Remark 4.7. Please note that there are no context rules for inputs or outputs, so that nothing happens under them.

The π-calculus gives rise to a transition monad as follows.

Placetaker types We consider two placetaker types, one for channels and one for processes. Hence, $\mathbb{P} = 2 = \{\mathbf{c}, \mathbf{p}\}$.

Placetaker monad Then, the syntax may be viewed as a monad $T : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{P}}$: given a variables object $X = (X_{\mathbf{c}}, X_{\mathbf{p}}) \in \mathbf{Set}^{\mathbb{P}}$, the placetaker object $T(X) = (X_{\mathbf{c}}, T(X)_{\mathbf{p}}) \in \mathbf{Set}^{\mathbb{P}}$ consists of the sets of channels and processes with free variables in X (modulo $\equiv$). Note that $T(X)_{\mathbf{c}} = X_{\mathbf{c}}$ as there is no operation on channels.

Transition type and state functors Transitions relate processes, so we take $\mathbb{S} = 1$ and $S_1(X) = S_2(X) = X_{\mathbf{p}}$.

Transition module Transitions are generated by the above three rules, and obviously stable under substitution. We thus obtain a transition monad.

Let us now describe a second way of organising the π-calculus as a transition monad, this time over a single placetaker type.

Placetaker Since all that needs substitution in the π-calculus is channels, we set $\mathbb{P} = 1$.

Placetaker monad Since the syntax contains no channel constructor, the placetaker monad is merely the identity monad.

Transition types and state functors However, since transitions relate processes, we need to fit the syntax into the state functors. We thus take $\mathbb{S} = 1$ and $S_1(X) = S_2(X)$ to be the set of processes with free channels in X.

Transition module Transitions are generated as above, and stable under channel renaming, hence again form a transition monad.

Remark 4.8. This would not work in the presence of the mismatch operator [57, p13], which breaks stability of reduction under renaming.

4.4. **Positive GSOS systems.** An example involving labelled transitions is given by Positive GSOS systems [19, p246]. They specify labelled transition systems, in which transitions have the shape $e \xrightarrow{a} f$, where a is drawn from some fixed set $\mathbb{A}$ of **labels**. Let us further fix a set O of **operations** with arities in $\mathbb{N}$ and an infinite set of **variables**, ranged over by x, y,...

Definition 4.9. A **Positive GSOS rule** has the shape

$$\frac{\dots \quad x_i \xrightarrow{a_{i,j}} y_{i,j} \quad \dots \quad (i \in n, j \in n_i)}{op(x_1, \dots, x_n) \xrightarrow{c} e}, \tag{4.1}$$

where

- $op \in O$ is an operation with arity n,
- for all $i \in n$, n_i is a natural number,
- the variables x_i and $y_{i,j}$ are all distinct, and
- e is an expression potentially depending on all the variables.

A **Positive GSOS system** is a set of Positive GSOS rules.

The semantics of a Positive GSOS rule is that of a "rule scheme", in the following sense.

Definition 4.10. The labelled transition relation **generated** by a Positive GSOS system is the smallest $\mathbb{A}$-labelled transition system on expressions generated by O, such that for all rules (4.1), and expressions $e_1, \dots, e_n$ and $e_{i,1}, \dots, e_{i,n_i}$ for all $i \in n$, if $e_i \xrightarrow{a_{i,j}} e_{i,j}$ for all $i \in n$ and $j \in n_i$, then

$$op(e_1, \dots, e_n) \xrightarrow{c} e[\dots, x_i \mapsto e_i, \dots, y_{i,j} \mapsto e_{i,j}, \dots].$$

Otherwise said, each rule scheme (4.1) induces a rule

$$\frac{\dots \quad e_i \xrightarrow{a_{i,j}} e_{i,j} \quad \dots (i \in n, j \in n_i)}{op(e_1, \dots, e_n) \xrightarrow{c} e[\dots, x_i \mapsto e_i, \dots, y_{i,j} \mapsto e_{i,j}, \dots]}.$$

Remark 4.11. The general notion of GSOS system includes negative rules, which means rules that may have premises of the shape $x_i \xrightarrow{a} \not\rightarrow$. Their semantics is significantly more involved, so we leave their integration as an open problem.

Each Positive GSOS system yields a transition monad as follows.

Placetaker and transition types We take $\mathbb{P} = 1$, because we are in an untyped setting, and $\mathbb{S} = 1$ because states are terms.

Placetaker monad The selected family of operations (or rather arities) specifies the term monad T.

State functors In order to take labels into account, we take $S_1(X) = X$ and $S_2(X) = \mathbb{A} \times X$. Transitions thus form a set over $X \times (\mathbb{A} \times X)$ as desired.

Remark 4.12. We could as well take $S_1(X) = \mathbb{A} \times X$ and $S_2(X) = X$, as in the end, only the product $S_1(X) \times S_2(X)$ matters.

Transition structure As before, we take as transitions the set of all proofs generated by the rules, which is indeed stable under substitution by construction.

4.5. **The differential λ-calculus.** Let us finally sketch how the differential λ-calculus [25] provides a further example with $S_1 \neq S_2$. Following Vaux [61, §6], its syntax may be defined by

$$\begin{array}{rcll} e, f, g & ::= & x \mid \lambda x.e \mid e\langle U\rangle \mid De \cdot f & \text{(terms)} \\ U, V & ::= & 0 \mid e + U & \text{(multiterms)}, \end{array}$$

where terms and multiterms are considered equivalent up to the following equations.

$$e + e' + U = e' + e + U \qquad\qquad D(De \cdot f) \cdot g = D(De \cdot g) \cdot f.$$

The definition of transitions is based on two auxiliary constructions:

(1) **Unary multiterm substitution** $e[x \mapsto U]$ of a multiterm U for a variable x in a term e, which returns a multiterm (not to be confused with unary monadic substitution, which handles the particular case where U is a mere singleton).
(2) **Partial derivative** $\frac{\partial e}{\partial x} \cdot U$ of a term e w.r.t. a term variable x along a multiterm U. This again returns a multiterm.

Both are defined by induction on e (see [61, Definition 6.3 and 6.4]).

We may now define the transition relation as the smallest context-closed relation satisfying the rules below.

$$(\lambda x.e)\langle U\rangle \to e[x \mapsto U] \qquad\qquad D(\lambda x.e) \cdot f \to \lambda x.\left(\frac{\partial e}{\partial x} \cdot f\right)$$

The compact formulation of the second rule relies on the abbreviation $\lambda x.(e_1 + \ldots + e_n) := \lambda x.e_1 + \ldots + \lambda x.e_n$.

Let us now sketch how this forms a transition monad.

Placetaker monad Terms induce a monad T on **Set**, which we take as the placetaker monad (hence $\mathbb{P} = 1$).

State functors Transitions relate terms to multiterms, hence $\mathbb{S} = 1$, S_1 is the identity, and $S_2 = !$ is the functor mapping any set X to the set of (finite) multisets over X.

Transition structure Transitions are stable under substitution by terms, hence we again have a transition monad.

5. Signatures for transition monads

In the previous section, we have shown that several significant (abstract) programming languages may be organised as transition monads. We are now interested in specifying such languages by signatures. We first introduce registers, as announced in the introduction, and then our register for transition monads.

5.1. **Registers of signatures.** In this section, we introduce registers of signatures, which are a formalisation of the notion of signature, at least in the context of initial algebra semantics.

The idea is that each signature S over a fixed category $\mathbf{C}$ should give rise to a particular category S-alg, equipped with a "forgetful" functor to $\mathbf{C}$, the specified object $spec(S)$ being the initial object of S-alg:

Definition 5.1. An abstract signature over a category $\mathbf{C}$ consists of a category $\mathbf{E}$ with an initial object, equipped with a functor $\mathbf{E} \to \mathbf{C}$. We denote by $\mathbf{SemSig}_{\mathbf{C}}$ the class of abstract signatures over $\mathbf{C}$.

Notation 5.2. *We denote the components of any abstract signature S over a category $\mathbf{C}$ by S-alg and $\mathbf{U}_S$, so that S is precisely $\mathbf{U}_S \colon S\text{-alg} \to \mathbf{C}$. Accordingly, we call objects of S-alg* S**-algebras**, *or* **models** *of S.*

Definition 5.3. A **register R** for a given category $\mathbf{C}$ consists of

- a class $\mathbf{Sig_R}$ of **signatures**, and
- a **semantics** map $[\![-]\!]_{\mathbf{R}} \colon \mathbf{Sig_R} \to \mathbf{SemSig_C}$.

Terminology 5.4. We say that any $S \in \mathbf{Sig_R}$ is a **signature for** $spec(S) := \mathbf{U}_S(0)$ (0 here denotes the initial object in S-alg), or alternatively that S **specifies** $spec(S)$. Finally, when we define our registers below, we first introduce signatures and associate a functor $\mathbf{E} \to \mathbf{C}$ to each signature. It then remains to prove that $\mathbf{E}$ has an initial object: as mentioned in §1, we call such proofs **validity proofs**.

Most of our registers will be monadic in the following sense.

Definition 5.5. A register $\mathbf{R}$ is **monadic** when the abstract signature $\mathbf{E} \to \mathbf{C}$ associated to any signature in $\mathbf{Sig_R}$ is finitary and monadic.

Remark 5.6. Strictly speaking, we should call this "finitary monadic". We omit the "finitary" for readability.

Notation 5.7. *When a register is monadic, we denote by $S^\star$ the monad induced by any signature S, in the sense that we have an isomorphism S-alg $\cong S^\star$-alg of categories over $\mathbf{C}$ (i.e., which commutes with forgetful functors). In such cases we have*

$$S^\star(0_{\mathbf{C}}) \cong spec(S) = \mathbf{U}_S(0_{S\text{-alg}}).$$

Let us now introduce a simple notion of morphism between registers.

Definition 5.8. A *compilation* from a register R_1 on a category $\mathbf{C}$ to a register R_2 on the same category is a map $c \colon \mathbf{Sig}_{R_1} \to \mathbf{Sig}_{R_2}$ preserving the semantics up to isomorphism, in the sense that for any $\Sigma \in \mathbf{Sig}_{R_1}$, there is an isomorphism $[\![\Sigma]\!]_{R_1} \cong [\![c(\Sigma)]\!]_{R_2}$ as objects of $\mathbf{CAT/C}$. We say that R_1 is a **subregister** of R_2 if there exists a compilation of R_1 to R_2.

Let us finish this subsection by recasting a well-known fact as the definition of a register. The well-known fact is the following.

Proposition 5.9 ([56, p62])**.** *For any finitary endofunctor F on a cocomplete category $\mathbf{C}$, the forgetful functor F-alg $\to \mathbf{C}$ is monadic, and the left adjoint maps any object $C \in \mathbf{C}$ to the initial algebra $F^*(C)$ of the functor $A \mapsto C+F(A)$, i.e., the least fixed point $\mu A.(C+F(A))$, with F-algebra structure given by*

$$F(F^*(C)) \hookrightarrow C + F(F^*(C)) \cong F^*(C).$$

And here comes the register:

Definition 5.10. For a given cocomplete category $\mathbf{C}$, we define the monadic register $\mathbf{EF_C}$, called the **endofunctor register**, as follows.

Signatures: A signature is a finitary endofunctor on $\mathbf{C}$.

Semantics: The abstract signature associated to any finitary endofunctor F is the forgetful functor $U_F \colon F\text{-alg} \to \mathbf{C}$.

Validity proof: By Proposition 5.9. □

Remark 5.11. Let F be any finitary endofunctor on a cocomplete category $\mathbf{C}$. Please note the difference between $F^\star(0)$ and $F^*(0)$: $F^\star(0)$ denotes the object specified by F *qua* signature of $\mathbf{EF_C}$, while $F^*(0)$ denotes the (carrier of the) initial F-algebra. In this case, of course, the denotations coincide, but this will no longer be the case, for instance, in §7.4. There, $F^\star(0)$ will denote the initial F-**monoid**, while $F^*(0)$ will still denote the initial F-algebra.

Let us conclude by naming all registers defined by compilation into $\mathbf{EF_C}$.

Definition 5.12. We call **endofunctorial** all subregisters of $\mathbf{EF_C}$.

5.2. **A register for transition monads.** So our goal is to define a register for transition monads. Thus, we should at least organise them into a category in the first place:

Definition 5.13. For any sets $\mathbb{P}$ and $\mathbb{S}$, finitary monad T over $\mathbf{Set}^{\mathbb{P}}$, and finitary functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, let

$$\mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) = T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/S_1T \times S_2T$$

denote the slice of the category of finitary, $\mathbf{Set}^{\mathbb{S}}$-valued T-modules over $S_1T \times S_2T$.

And for any sets $\mathbb{P}$ and $\mathbb{S}$, we let $\mathbf{TransMnd}_{\mathbb{P},\mathbb{S}} := \sum_{T,S_1,S_2} \mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$.

Remark 5.14. Reduction monads of [9] correspond to the case when $S_1 = S_2 = \mathrm{Id}$, with $\mathbb{P} = \mathbb{S} = 1$, but contrary to the present work, morphisms there can live between reduction monads with different underlying monads. We don't need such morphisms in the present work because we enforce that models of a signature share the same underlying monad. This allows for a simpler notion of signature, at the cost of reducing the scope of the recursion principle.

In the coming sections, we will introduce

- a register $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$ for finitary monads on the category $\mathbf{Set}^{\mathbb{P}}$,
- a register $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ for finitary functors $\mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, and
- given a finitary monad T and finitary functors S_1 and S_2, a register $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ for transition structures over T, S_1, and S_2.

Assuming this is done, we may already give our register for transition monads:

Definition 5.15. We define the register $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$ for the category $\mathbf{TransMnd}_{\mathbb{P},\mathbb{S}}$ as follows.

Signatures: A signature, which we call a **transition signature**, consists of

- a signature Σ of $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$, specifying a finitary monad T on $\mathbf{Set}^{\mathbb{P}}$,
- signatures Σ_1 and Σ_2 of $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$, specifying functors $S_1, S_2 \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$,
- a signature Σ_{Trans} of $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$.

Semantics: The abstract signature associated to a signature $(\Sigma, \Sigma_1, \Sigma_2, \Sigma_{Trans})$ is

$$\Sigma_{Trans}\text{-alg} \xrightarrow{\mathbf{U}_{\Sigma_{Trans}}} \mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) \hookrightarrow \mathbf{TransMnd}_{\mathbb{P},\mathbb{S}}.$$

In order for this definition — and with it all register definitions — to make sense, we should prove that the semantics map does indeed return abstract signatures.

Validity proof: We need to prove that Σ_{Trans}-alg has an initial object; but this follows from $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ being a register, which will be proved below. □

Main Register	Signatures	Base category	Where
$\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$	Transition signature	$\mathbf{TransMnd}_{\mathbb{P},\mathbb{S}}$	5.15
$\mathbf{RegITransMnd}_{\mathbb{P},\mathbb{S}}$	*idem*	$\mathbf{ITransMnd}_{\mathbb{P},\mathbb{S}}$	6.21
$\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T,S_1,S_2)$	Families of rules	$\mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T,S_1,S_2)$	6.14
$\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$	Equational modular signatures	$\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$	7.15
$\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{S}}]_f$	Equational facet-based signatures	$[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{S}}]_f$	8.14
Auxiliary Register	Signatures	Base category	Where
$\mathbf{R}^*$	Families of signatures of $\mathbf{R}$	Same as $\mathbf{R}$	6.13
$\mathbf{RegTransStruct}^0_{\mathbb{P},\mathbb{S}}(T,S_1,S_2)$	Rules	$\mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T,S_1,S_2)$	6.12
$\mathbf{Reg}^0(\mathbf{C}/X)$	Abstract simple rules	$\mathbf{C}/X$	6.2
$\mathbf{Reg}^1(\mathbf{C}/X)$	Abstract medium rules	$\mathbf{C}/X$	6.5
$\mathbf{Reg}(\mathbf{C}/X)$	Abstract rules	$\mathbf{C}/X$	6.9
$\mathbf{RegMnd}^0_f(\mathbf{Set}^{\mathbb{P}})$	Modular signatures	$\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$	7.11
$\mathbf{Reg}^0[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{S}}]_f$	Facet-Based signatures	$[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{S}}]_f$	8.11
General Register	Signatures	Base category	Where
$\mathbf{EF}_{\mathbf{C}}$	Finitary endofunctors	$\mathbf{C}$	5.10
$\mathbf{ES}_{\mathbf{C}}$	Equational systems	$\mathbf{C}$	9.11
$\mathbf{PSEF}_{\mathbf{C}}$	Pointed strong endofunctors	$\mathbf{Mon}(\mathbf{C})$	9.22
$\mathbf{MES}_{\mathbf{C}}$	Monoidal equational systems	$\mathbf{Mon}(\mathbf{C})$	9.29

For $\mathbf{R}^*$, $\mathbf{R}$ is assumed to be an endofunctorial register (Definition 5.12).

Figure 1: Registers

It remains to introduce the registers $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$, $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{S}}]_f$, and $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T,S_1,S_2)$. The most novel among these is clearly the latter. It is furthermore independent from the others, so we introduce it first.

For the reader's convenience, we list our registers in Figure 1, together with corresponding categories.

6. Registers for transition structures

In this section, we define the register $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T,S_1,S_2)$ for transition structures on fixed T,S_1,S_2. Since these are $\mathbf{Set}^{\mathbb{S}}$-valued T-modules over $S_1T \times S_2T$, this register should specify objects of the slice category $T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/S_1T\times S_2T$. We will in fact design a register for more general slice categories $\mathbf{C}/X$, where $\mathbf{C}$ is a locally finitely presentable category and X an object of $\mathbf{C}$. The desired register $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T,S_1,S_2)$ will then be obtained as an instance by taking $\mathbf{C} = T\text{-}\mathbf{Mod}_f$ and $X = S_1T \times S_2T$.

In §6.1, we first present a basic register $\mathbf{Reg}^0(\mathbf{C}/X)$ that would work for the untyped case without variable binding. Then, in §6.2, we extend $\mathbf{Reg}^0(\mathbf{C}/X)$ to a more powerful register $\mathbf{Reg}^1(\mathbf{C}/X)$ that deals with variable binding. Observing that this register is slightly heavy to use in the typed setting, we design a more convenient variant, called $\mathbf{Reg}(\mathbf{C}/X)$. Finally, in §6.4, we focus on instances of this register to categories of the form $\mathbf{C} = T\text{-}\mathbf{Mod}_f/(S_1 \circ T \times S_2 \circ T)$. In this case, we introduce special notation, which allows us to write signatures essentially as in usual operational semantics literature. Finally, in §6.5, we derive a register for the proof-irrelevant variant of transition monads.

6.1. **Small register for slice categories.** In this section, we introduce a first, limited register for slice categories.

Example 6.1. Let $\mathbb{P}$ denote the set of simple types over a given set of basic types, as in §4.1, and consider the arity for application, i.e., the endofunctor $\Sigma_{\mathrm{app}} \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{P}}$ defined by

$$\Sigma_{\mathrm{app}}(X)(A) = \sum_B X(B \to A) \times X(B).$$

Across the equivalence $\mathbf{Set}^{\mathbb{P}} \simeq \mathbf{Set}/\mathbb{P}$, one way of presenting this endofunctor which is perhaps closer to the syntactic inference rule for application is as the span

$$\mathbb{P}^2 \xleftarrow{\langle \mathrm{arr}, \pi_2 \rangle} \mathbb{P}^2 \xrightarrow{\pi_1} \mathbb{P},$$

where $\mathrm{arr}(A, B) = (B \to A)$. Indeed, $\Sigma_{\mathrm{app}}(X)$ corresponds to

- taking the product of $X \to \mathbb{P}$ with itself in the arrow category,
- pulling back along $\langle \mathrm{arr}, \pi_2 \rangle$, and
- postcomposing with π_1.

To see this, let us observe that after pulling back, we obtain a family X' over $\mathbb{P}^2$ such that $X'(A, B) = X(B \to A) \times X(B)$. Postcomposing, we take the disjoint union over A as desired.

Generalising from this example, we obtain the following "small" register.

Definition 6.2. For any locally finitely presentable category $\mathbf{C}$ and object $X \in \mathbf{C}$, we define the endofunctorial register $\mathbf{Reg}^0(\mathbf{C}/X)$ for the slice category $\mathbf{C}/X$ as follows.

Signatures: A signature consists of:

- a **metavariable** object V;
- a list of **premise** morphisms $(V \xrightarrow{s_i} X)_{i \in n}$ denoted by $V \xrightarrow{\vec{s}} X^n$;
- a **conclusion** morphism $V \xrightarrow{t} X$.

Semantics: A model of a signature $(X^n \xleftarrow{\vec{s}} V \xrightarrow{t} X)$ is an algebra for the endofunctor mapping any $Y \xrightarrow{p} X$ to $Y' \to V \xrightarrow{t} X$, where $Y' \to V$ denotes the following pullback.

$$\begin{array}{ccc} Y' & \longrightarrow & Y^n \\ \downarrow & \lrcorner & \downarrow{\scriptstyle p^n} \\ V & \xrightarrow[\vec{s}]{} & X^n \end{array}$$

Morphisms of models are morphisms of algebras.

Validity proof: All we have to show is that the induced endofunctor is finitary. But this functor is a composite of three functors

$$\mathbf{C}/X \xrightarrow{(-)^n/X} \mathbf{C}/X^n \xrightarrow{\vec{s}^*} \mathbf{C}/V \xrightarrow{t_!} \mathbf{C}/X,$$

where

- $(-)^n/X$ denotes n-fold self-product in the arrow category,
- $\vec{s}^*$ denotes pullback along $\vec{s}$, and
- $t_!$ denotes postcomposition with t.

The last two functors, as left adjoints, are cocontinuous, hence finitary. Finally, the forgetful functor $\mathbf{C}/X^n \to \mathbf{C}$ creates colimits, so it suffices to show that the composite functor $\mathbf{C}/X \xrightarrow{(-)^n/X} \mathbf{C}/X^n \to \mathbf{C}$ is finitary. But this in turn is the composite of $\mathbf{C}/X \to \mathbf{C} \xrightarrow{(-)^n} \mathbf{C}$. Because the first component creates, hence preserves all colimits, it suffices to show that $(-)^n$ is finitary, which holds since filtered colimits commute with finite limits in locally finitely presentable categories [3, Proposition 1.59]. □

Example 6.3. Let $\mathbb{P} = 1$ so that $\mathbf{Set}^{\mathbb{P}} \cong \mathbf{Set}$, and T denote the monad for pure λ-calculus syntax. Taking $\mathbf{C} = T\text{-}\mathbf{Mod}_f(\mathbf{Set})$ and X the product module $T^2 = T \times T$, let us consider the left congruence rule for application

$$\frac{M \longrightarrow N}{M\ P \longrightarrow N\ P}$$

as a signature of $\mathbf{Reg}^0(\mathbf{C}/X)$. For this, we take

- as metavariable module $V = T^3$,
- a single premise given by $\langle \pi_1, \pi_2 \rangle \colon T^3 \to T^2$, and
- as conclusion the morphism $t \colon T^3 \to T^2$ mapping any (M, N, P) to $(M\ P, N\ P)$.

Let us now see what it means for a module morphism $R \to T^2$ to be a model. In this case, the pullback along $\langle \pi_1, \pi_2 \rangle$ yields $R \times T$, and so a model structure amounts to a morphism $R \times T \to R$ making the following square commute.

$$\begin{array}{ccc} R \times T & \longrightarrow & R \\ \downarrow & & \downarrow \\ T^2 \times T & \xrightarrow[t]{} & T^2 \end{array}$$

Unfolding the definition, such a map associates to any transition $r \in R(X)$ over $(M, N) \in T^2(X)$ and term $P \in T(X)$ a transition over $t(M, N, P) = (M\ P, N\ P)$, as desired.

6.2. **Binding register for slice categories.** In this section, we observe that the register $\mathbf{Reg}^0(\mathbf{C}/X)$ is not expressive enough in the presence of variable binding, hence we proceed to define a refined variant.

Example 6.4. Let $\mathbb{P} = 1$ so that $\mathbf{Set}^{\mathbb{P}} \cong \mathbf{Set}$, and T denote the monad for pure λ-calculus syntax. Taking $\mathbf{C} = T\text{-}\mathbf{Mod}_f$ and $X = T^2$ as in Example 6.3, let us consider the ξ-rule

$$\frac{M \to N}{\lambda x.M \to \lambda x.N}\ .$$

The natural metavariable module for this is $T^{(1)} \times T^{(1)}$, because M and N have an additional, fresh variable, and the natural premise would be the identity map thereon. However, the register $\mathbf{Reg}^0(\mathbf{C}/X)$ only allows premises to target powers of X.

In order to remedy this situation, we refine $\mathbf{Reg}^0(\mathbf{C}/X)$ to obtain the following "medium" register.

Definition 6.5. For any locally finitely presentable category $\mathbf{C}$ and object $X \in \mathbf{C}$, we define the endofunctorial register $\mathbf{Reg}^1(\mathbf{C}/X)$ for the slice category $\mathbf{C}/X$ as follows.

Signatures: A signature consists of:

- a **metavariable** object V;
- a list of **premise** morphisms $(V \xrightarrow{s_i} F_i X)_{i\in n}$, where each F_i is a finitary endofunctor on $\mathbf{C}$, denoted by $V \xrightarrow{\vec{s}} \prod_i F_i X$;
- a **conclusion** morphism $V \xrightarrow{t} X$.

Semantics: A model of a signature $(\prod_i F_i X \xleftarrow{\vec{s}} V \xrightarrow{t} X)$ is an algebra for the functor mapping $Y \xrightarrow{p} X$ to $Y' \to V \xrightarrow{t} X$, where Y' denotes the pullback

$$\begin{array}{ccc} Y' & \longrightarrow & \prod_i F_i(Y) \\ \downarrow & & \downarrow{\scriptstyle \prod_i F_i(p)} \\ V & \xrightarrow[\vec{s}]{} & \prod_i F_i(X) \end{array}$$

Validity proof: Similar to Definition 6.2, with the following composite endofunctor.

$$\mathbf{C}/X \xrightarrow{(\prod_i F_i(-))/X} \mathbf{C}/\prod_i F_i(X) \xrightarrow{\vec{s}^*} \mathbf{C}/V \xrightarrow{t_!} \mathbf{C}/X$$ □

Example 6.6. Let us now treat the ξ-rule, rectifying Example 6.4.

- We first take as metavariable module $V := T^{(1)} \times T^{(1)}$, as planned.
- We then take as unique premise the identity on V. For this we should justify that V does have the desired form $F(T^2)$. This is the case for $F(M) = M^{(1)}$ since $(T^2)^{(1)} = (T^{(1)})^2$.
- Finally, we take as conclusion $V := T^{(1)} \times T^{(1)} \xrightarrow{\lambda\times\lambda} T \times T$.

Remark 6.7. We may generalise this register to permit a conclusion between terms with additional free variables, by having the conclusion morphism target RX rather than X, for some finitary right adjoint functor R. For example, exploiting the adjunction $-^{(1)} \vdash - \times T$ in the category of T-modules [7, Proposition 13], the application of untyped λ-calculus can be viewed as an operation $\mathrm{app}_{alt} : T \to T^{(1)}$. Anticipating on §6.4, the corresponding modified β-rule is

$$\frac{}{\mathrm{app}_{alt}(abs(t)) \rightsquigarrow t}\ .$$

The point here is that for any set X and $t \in T^{(1)}(X)$, both terms $\mathrm{app}_{alt}(abs(t))$ and t lie in $T^{(1)}(X)$.

6.3. **Typed variant.** In this section, we observe that the medium register $\mathbf{Reg}^1(\mathbf{C}/X)$ is slightly inconvenient in a typed setting, and propose our final register $\mathbf{Reg}(\mathbf{C}/X)$ for slice categories.

Example 6.8. Let $\mathbb{P}$ denote the set of simple types over a given set of ground types and $T : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{P}}$ denote the monad for simply-typed λ-calculus values, and let $S_1 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{P}}$ denote the first state functor from §4.1, for which $S_1(X)$ is the set of typed application binary trees with leaves in X. Let us consider the β-rule

$$\frac{e_1 \Downarrow \lambda(e_3) \qquad e_2 \Downarrow w \qquad e_3[w] \Downarrow v}{e_1\ e_2 \Downarrow v}\ .$$

Implicitly, this is in fact a family of rules indexed over all pairs $(A, B) \in \mathbb{P}^2$ of types. For any such (A, B), we have $e_1 \in S_1T(X)_{A\to B}$, $e_2 \in S_2T(X)_A$, and $e_1\ e_2 \in S_1T(X)_B$. But these are all **Set**-valued modules, while the transition module $R \to S_1T \times S_2T$ is $\mathbf{Set}^{\mathbb{P}}$-valued.

In order to work with **Set**-valued modules, we now want to introduce a refinement of the register $\mathbf{Reg}^1(\mathbf{C}/X)$. Let us first sketch on this example how it should look like. First of all, because $e_1\ e_2$ and v have type B, we would like to replace the $\mathbf{Set}^{\mathbb{P}}$-valued module $S_1T \times S_2T$ with the **Set**-valued $(S_1T \times S_2T)_B$. We would then use as metavariable module the product

$$V = (S_1T)_{A\to B} \times (S_1T)_A \times (S_1T)_B^{(A)} \times T_B \times T_A,$$

whose elements are tuples (e_1, e_2, e_3, v, w) as in the rule. The conclusion of our rule should then consist of a morphism $V \to (S_1T \times S_2T)_B$, and similarly for the premises (see Example 6.11 below). The crucial ingredient here is the functor $(-)_B \colon T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}}) \to T\text{-}\mathbf{Mod}_f(\mathbf{Set})$. Furthermore, in order to prove the existence of an initial model, it is important that this functor has a right adjoint $(-) \cdot \mathbf{y}_B$, as in (7.1).

Abstracting over this situation, we are led to:

Definition 6.9. For any locally finitely presentable category $\mathbf{C}$ and object $X \in \mathbf{C}$, we define the endofunctorial register $\mathbf{Reg}(\mathbf{C}/X)$ for the slice category $\mathbf{C}/X$ as follows.

Signatures: A signature, called a **rule**, consists of:

- a category $\mathbf{D}$ and a right adjoint $E \colon \mathbf{C} \to \mathbf{D}$;
- a **metavariable** object $V \in \mathbf{D}$;
- a list of **premise** morphisms $(V \xrightarrow{s_i} F_iX)_{i\in n}$, where each $F_i \colon \mathbf{C} \to \mathbf{D}$ is a finitary functor, denoted by $V \xrightarrow{\vec{s}} \prod_i F_iX$;
- a **conclusion** morphism $V \xrightarrow{t} E(X)$.

Semantics: Let us consider any signature S, consisting of $E \colon \mathbf{C} \to \mathbf{D}$, with left adjoint $J \colon \mathbf{D} \to \mathbf{C}$, and morphisms $(\prod_i F_iX \xleftarrow{\vec{s}} V \xrightarrow{t} E(X))$. Then, S induces a functor $\Sigma_S \colon \mathbf{C}/X \to \mathbf{D}/E(X)$ mapping any $Y \xrightarrow{p} X$ to $Y' \to V \xrightarrow{t} E(X)$, where Y' denotes the following pullback.

$$\begin{array}{ccc} Y' & \longrightarrow & \prod_i F_i(Y) \\ \downarrow & & \downarrow{\scriptstyle \prod_i F_i(p)} \\ V & \xrightarrow[\vec{s}]{} & \prod_i F_i(X) \end{array}$$

Composing with transposition $\widetilde{(-)} \colon \mathbf{D}/E(X) \to \mathbf{C}/X$, we obtain an endofunctor $\widetilde{\Sigma_S}$, and define the abstract signature associated to S to be the forgetful functor

$$\widetilde{\Sigma_S}\text{-alg} \to \mathbf{C}/X.$$

Remark 6.10. Equivalently, a model is a morphism $p \colon Y \to X$ equipped with a map k making the following triangle commute,

$$\begin{array}{ccccc} Y' & & \xrightarrow{\quad k \quad} & & E(Y) \\ & {\scriptstyle \Sigma_S(Y,p)} \searrow & & \swarrow {\scriptstyle E(p)} & \\ & & E(X) & & \end{array}$$

and a morphism of models $(Y, p) \to (Z, q)$ is a morphism f in $\mathbf{C}/X$ making the following square commute.

$$\begin{array}{ccc} Y' & \xrightarrow{k_Y} & E(Y) \\ {\scriptstyle f'}\downarrow & & \downarrow{\scriptstyle E(f)} \\ Z' & \xrightarrow[k_Z]{} & E(Z) \end{array}$$

Validity proof: The induced functor Σ_S is a left adjoint, hence cocontinuous. Furthermore, the transposition functor $\widetilde{=}$ is cocontinuous, because colimits in slice categories are computed on domains, on which it acts like the left adjoint J. The composite endofunctorial is thus finitary, as desired. □

Example 6.11. Let us now treat the big-step β-rule, rectifying Example 6.8. For any types A and B, we define the following rule:

- we take $\mathbf{D} = T\text{-}\mathbf{Mod}_f(\mathbf{Set})$ and $E : T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}}) \to T\text{-}\mathbf{Mod}_f(\mathbf{Set})$ to be pointwise evaluation at B, which is indeed right adjoint to $(-) \cdot \mathbf{y}_B$ (see (7.1) below);
- we further take $V := (S_1T)_{A\to B} \times (S_1T)_A \times (S_1T)_B^{(A)} \times T_B \times T_A$ as metavariable object;
- we have three premises $V \to S_1T \times S_2T$, which, at any $X \in \mathbf{Set}^{\mathbb{P}}$, respectively map any $(e_1, e_2, e_2, w, v) \in V(X)$ to:
  - $(e_1, \lambda_{A,B}(e_3))$,
  - (e_2, w), and
  - $(e_3[w], v)$;
- the conclusion maps any such $(e_1, e_2, e_2, w, v) \in V(X)$ to $(\mathrm{app}_{A,B}(e_1, e_2), v)$.

Using the notation of §6.4 below, this will look much like the standard, syntactic rule.

Definition 6.12. Let $\mathbf{RegTransStruct}^0_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) = \mathbf{Reg}(T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/S_1T \times S_2T)$.

Finally, we would like signatures to consist of families of rules. For this, we use the following generic construction of registers.

Definition 6.13. For any endofunctorial register $\mathbf{R}$ for a category with coproducts, we denote by $\mathbf{R}^*$ the endofunctorial register whose signatures are families of signatures in $\mathbf{Sig}_{\mathbf{R}}$, and whose semantics maps any family to the coproduct of associated endofunctors.

We may at last define our register for the category $\mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ of transition structures, which we recall is by definition the slice category $T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/S_1T \times S_2T$. For this, we take as signatures all families of signatures in $\mathbf{RegTransStruct}^0_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$:

Definition 6.14. Let $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) = \mathbf{RegTransStruct}^0_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)^*$.

6.4. **A format for displaying signatures in rule-based registers.** In all of our examples of signatures in $\mathbf{RegTransStruct}^0_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$, the metavariable object V is a functor to $\mathbf{Set}$, so the premises and conclusion are set-maps (which are in fact module morphisms). In this case, we adopt the following notational conventions.

- For each premise or conclusion $\begin{array}{ccc} V & \to & W \\ x & \mapsto & E \end{array}$ of a rule, we write $x : V \vdash E : W$.
- Furthermore, we organise the premises and conclusion as usual:

$$\frac{x\colon V \vdash E_1\colon W_1 \qquad \dots \qquad x\colon V \vdash E_n\colon W_n}{x\colon V \vdash E\colon W},$$

or just $\dfrac{E_1 \qquad \dots \qquad E_n}{E}$ when the rest may be inferred from context.

Moreover, in our examples $M = S_1T \times S_2T$, so each element E is in fact a pair (L, R), which we generally denote with an arrow, e.g., by $L \rightsquigarrow R$, $L \to R$,....

Example 6.15. The big-step β-rule from Example 6.11 reads as follows.

$$\frac{e_1 \rightsquigarrow \lambda_{A,B}(e_3) \qquad e_2 \rightsquigarrow w \qquad e_3[w] \rightsquigarrow v}{\mathrm{app}_{A,B}(e_1, e_2) \rightsquigarrow v}$$

Remark 6.16. The module V is often a product and thus x is a tuple.

Remark 6.17 ([9])**.** In practice, there are several choices for building the transition rule out of such a schematic presentation, depending on the order of metavariables. This order is irrelevant: all interpretations yield isomorphic semantics, in the obvious sense.

Remark 6.18. This format could be generalised to any metavariable object, by using the internal language of categories.

6.5. **Proof-irrelevant variant.** In this section, we introduce a register for the proof-relevant variant of transition monads. The idea is very simple: we keep the same signatures as in the proof-relevant setting, and interpret each signature in a proof-irrelevant way. This is done by constructing a functor from proof-relevant transition monads to proof-irrelevant ones.

Proposition 6.19. *Let* $\mathbf{ITransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ *denote the full subcategory of transition structures* $\langle src, tgt\rangle : R \to S_1T \times S_2T$ *such that* $\langle src, tgt\rangle$ *is a pointwise inclusion. Then, the embedding* $U_{T,S_1,S_2} : \mathbf{ITransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) \hookrightarrow \mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ *is reflective. Consequently, letting* $\mathbf{ITransMnd}_{\mathbb{P},\mathbb{S}} := \sum_{T,S_1,S_2} \mathbf{ITransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$*, the induced embedding* $\mathbf{ITransMnd}_{\mathbb{P},\mathbb{S}} \hookrightarrow \mathbf{TransMnd}_{\mathbb{P},\mathbb{S}}$ *is also reflective.*

Proof. The left adjoint $L : \mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) \to \mathbf{ITransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ maps a transition structure $\partial : R \to S_1T \times S_2T$ to the monomorphism $\overline{R} \hookrightarrow S_1T \times S_2T$ obtained from the (strong epi)-mono factorisation[2] of ∂. Then, the natural bijection

$$\mathbf{TransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)(T_1, UT_2) \simeq \mathbf{ITransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)(LT_1, T_2)$$

follows from the lifting property of strong epimorphisms. □

Let us now introduce the relevant register. For this, we first observe that postcomposition with a functor $F\colon \mathbf{C} \to \mathbf{D}$ turns a register for $\mathbf{C}$ into one for $\mathbf{D}$.

Definition 6.20 (Post-composition register)**.** For any functor $F\colon \mathbf{C} \to \mathbf{D}$ and register $\mathbf{R}$ on $\mathbf{C}$, let $F_!(\mathbf{R})$ denote the register for $\mathbf{D}$ with $\mathbf{Sig}_{F_!(\mathbf{R})} = \mathbf{Sig}_{\mathbf{R}}$ and $[\![s]\!]_{F_!(\mathbf{R})} = F \circ [\![s]\!]_{\mathbf{R}}$.

Definition 6.21. The register $\mathbf{RegITransMnd}_{\mathbb{P},\mathbb{S}}$ is defined as $F_!(\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}})$, where $F\colon \mathbf{TransMnd}_{\mathbb{P},\mathbb{S}} \to \mathbf{ITransMnd}_{\mathbb{P},\mathbb{S}}$ denotes the reflection.

[2]As mentioned in Remark 3.8, the category of finitary **Set**-valued T-modules is a presheaf category, and thus has (strong epi)-mono factorisations.

7. Registers for monads

In this section and the next, we design the missing registers, respectively for monads and state functors. The registers are mostly adapted from existing constructions and results in the literature [8, 29, 27, 28]. The novelty here mainly lies in our new explicit description of initial algebras.

The basic idea for specifying operations in our register for monads is that the arity of an operation consists of

- an "input" (**Set**-valued) **parametric** module, in the sense of module that is definable for any monad on $\mathbf{Set}^{\mathbb{P}}$, together with
- an "output" placetaker type $p \in \mathbb{P}$.

The role of parametric modules lies in specifying how capture-avoiding substitution should interact with operations. A similar role is played in [29, 27] by "pointed strong" endofunctors; we explain the connection in §9.2.

In §7.1, we introduce the **D**-valued parametric modules announced in §2.4. In §7.2, we construct a first register $\mathbf{RegMnd}_f^0(\mathbf{Set}^{\mathbb{P}})$, which only allows to specify operations. We then deal with equations in §7.3. Finally, we characterise initial algebras in §7.4. All proofs are deferred to §10.

7.1. Parametric modules.

We start by reviewing some categories of modules.

Definition 7.1. Given categories **C** and **D**, let $\mathbf{Mod}(\mathbf{C},\mathbf{D})$, or $\mathbf{Mod}(\mathbf{D})$ when **C** is clear from context, denote the category

- whose objects are pairs (T, M) of a finitary monad T on **C** and a finitary T-module $M\colon \mathbf{C} \to \mathbf{D}$,
- and whose morphisms $(T, M) \to (U, N)$ are pairs (α, β) of a monad morphism $\alpha\colon T \to U$ and a natural transformation $\beta\colon M \to N$ commuting with action, in the sense that the following square commutes.

$$\begin{array}{ccc} M \circ T & \xrightarrow{\beta\circ\alpha} & N \circ U \\ \downarrow & & \downarrow \\ M & \xrightarrow[\beta]{} & N \end{array}$$

The first projection yields a forgetful functor $\mathbf{p}\colon \mathbf{Mod}(\mathbf{D}) \to \mathbf{Mnd}_f$.

Definition 7.2. A (**D**-valued) **parametric** module is a section of **p**, or in other words a functor $s\colon \mathbf{Mnd}_f \to \mathbf{Mod}(\mathbf{D})$ such that $\mathbf{p} \circ s = \mathrm{id}_{\mathbf{Mnd}_f}$.

Terminology 7.3. In the following, $\mathbf{C} = \mathbf{Set}^{\mathbb{P}}$, and parametric modules are implicitly **Set**-valued by default.

Example 7.4. Let us start by a few basic constructions of parametric modules:

- we denote by Θ the $\mathbf{Set}^{\mathbb{P}}$-valued parametric module mapping a monad T to itself, as a module over itself;
- for any $p_1, \ldots, p_n \in \mathbb{P}$ and **D**-valued parametric module M, let $M^{(p_1,\ldots,p_n)}$ associate to each monad T the T-module $M(T)^{(p_1,\ldots,p_n)}$ as in §2.4, i.e., $M^{(p_1,\ldots,p_n)}(T)(X) = M(T)(X + \mathbf{y}_{p_1} + \ldots + \mathbf{y}_{p_n})$; when $\mathbb{P} = 1$, we merely count the p_i's and write $M^{(n)}$;

- for any finitary functor $F\colon \mathbf{D} \to \mathbf{E}$ and $\mathbf{D}$-valued parametric module M, the $\mathbf{E}$-parametric module $F \circ M$ maps any monad T to the T-module $F \circ M(T)$; as particular cases:
  - when $\mathbf{D}$ has a terminal object, the terminal $\mathbf{D}$-valued parametric module $1 = 1 \circ \Theta$ maps any monad T to the constant T-module 1;
  - for any $p \in \mathbb{P}$ and $\mathbf{Set}^{\mathbb{P}}$-valued parametric module M, we denote by M_p the $\mathbf{Set}$-valued parametric module mapping any monad T to the T-module $X \mapsto M(X)_p$ (this is used extensively below, notably in §11);
  - in particular, for any $p \in \mathbb{P}$, the $\mathbf{Set}$-valued parametric module Θ_p maps a monad T on $\mathbf{Set}^{\mathbb{P}}$ to the module $X \mapsto T(X)_p$;
  - given a finite family $(M_i)_{i \in I}$ of $\mathbf{Set}$-valued parametric modules, I, let $\prod_i M_i$ associate to any monad T the T-module $\prod_i M_i(T)$.

Example 7.5. Recall that the idea of our register is that an operation will be specified by two parametric modules, one for the source and another (very simple) for the target. Let us give the parametric modules for a few operations from our examples.

Language	Operation	Source	Target
Pure $\overline{\lambda}\mu$	Push	$\Theta_{\mathbf{p}} \times \Theta_{\mathbf{s}}$	$\Theta_{\mathbf{s}}$
Pure $\overline{\lambda}\mu$	Abstraction	$\Theta_{\mathbf{p}}^{(\mathbf{p})}$	$\Theta_{\mathbf{p}}$
π-calculus	Input $a(b).P$	$\Theta_{\mathbf{c}} \times \Theta_{\mathbf{p}}^{(\mathbf{c})}$	$\Theta_{\mathbf{p}}$

In the above table, $\mathbf{p}$, $\mathbf{s}$, and $\mathbf{c}$ are placetaker types, and the subscripts on Θ refer to the notation M_p introduced in Example 7.4. Thus, e.g., $(\Theta_{\mathbf{p}} \times \Theta_{\mathbf{s}})(T)(X) = T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}}$.

In the following, we will at times use two distinct viewpoints on our signatures for monads. One, more user-friendly, is based on $\mathbf{Set}$-valued (= heterogeneous) parametric modules. The other, more efficient for stating the explicit description of initial algebras, is based on $\mathbf{Set}^{\mathbb{P}}$-valued (= homogeneous) modules. The two viewpoints are related by a $\mathbb{P}$-indexed family of adjunctions, which we now recall. For all $r \in \mathbb{P}$, there is an adjunction

$$[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}]_f \underset{(-)_r}{\overset{(-)\cdot \mathbf{y}_r}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f,$$

where

- $\mathbf{y}_r : \mathbf{Set}^{\mathbb{P}}$ denotes the family with a single element, sitting over $r \in \mathbb{P}$, and
- the left adjoint maps any $F\colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}$ to the endofunctor $(X \in \mathbf{Set}^{\mathbb{P}}) \mapsto F(X) \cdot \mathbf{y}_r$, i.e., $(F(X) \cdot \mathbf{y}_r)(r) = F(X)$ and $(F(X) \cdot \mathbf{y}_r)(p) = \emptyset$ when $p \neq r$.

Indeed, we have a natural isomorphism

$$\frac{F(X) \cdot \mathbf{y}_r \to G(X)}{F(X) \to G(X)(r)}.$$

Proposition 7.6. *These adjunctions lift to a $\mathbb{P}$-indexed family of adjunctions*

$$\mathbf{Mod}(\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}) \underset{(-)_r}{\overset{(-)\cdot \mathbf{y}_r}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{Mod}(\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}). \tag{7.1}$$

Proof. Straightforward. □

7.2. **The register $\mathbf{RegMnd}_f^0(\mathbf{Set}^{\mathbb{P}})$ for specifying operations.** This section is devoted to defining the monadic register $\mathbf{RegMnd}_f^0(\mathbf{Set}^{\mathbb{P}})$.

Signatures will rely on parametric modules, but these need to be restricted in order to ensure existence of an initial algebra (and even monadicity).

Proposition 7.7. *For any set $\mathbb{P}$, $p_1, \dots, p_n \in \mathbb{P}$, and finitary functor $F\colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}$, the assignment $T, X \mapsto F(T(X + \sum_{i\in n} \mathbf{y}_{p_i}))$ defines a parametric module denoted by $(F \circ \Theta)^{(p_1,\dots,p_n)}$.*

Proof. Straightforward. □

Definition 7.8. A parametric module is **elementary** if it is isomorphic to some finite product of parametric modules of the shape $(F \circ \Theta)^{(p_1,\dots,p_n)}$.

Example 7.9. Typically, taking $F(X) = X(p)$, any finite product of parametric modules of the shape $\Theta_p^{(p_1,\dots,p_n)}$, for some $p, p_1, \dots, p_n \in \mathbb{P}$, is elementary.

Definition 7.10. Given any set $\mathbb{P}$, a **modular signature** is a family of pairs (d, p) where

- d is an elementary parametric module, and
- $p \in \mathbb{P}$.

Given any modular signature $S = (d_i, p_i)_{i\in I}$, an **S-algebra** is a finitary monad T equipped with module morphisms $d_i(T) \to T_{p_i}$ for all $i \in I$. An S-algebra morphism is a monad morphism commuting with these morphisms. We denote by $S\text{-alg} \to \mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$ the forgetful functor.

Definition 7.11. We define the monadic register $\mathbf{RegMnd}_f^0(\mathbf{Set}^{\mathbb{P}})$ for $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$ as follows, for any set $\mathbb{P}$.

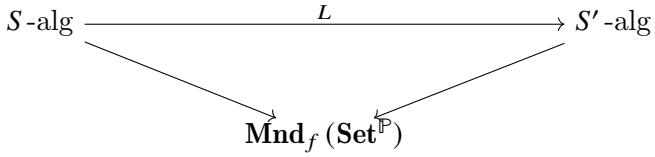
Signatures: A signature is a modular signature.

Semantics: The abstract signature associated to any signature S is the forgetful functor $S\text{-alg} \to \mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$.

Validity proof: By Corollary 10.21 below. □

7.3. **The register $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$.** We now define our register $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$, where a signature will consist of a signature of $\mathbf{RegMnd}_f^0(\mathbf{Set}^{\mathbb{P}})$, plus a family of "equations". An equation is essentially a pair of "derived operations" with a common "arity". The arity consists of an input arity, which intuitively models the metavariables of the equation, and an output arity, which models the output type. The input arity will be an elementary parametric module d, and the output arity will be a placetaker type $p \in \mathbb{P}$. For a family of equations, the arity thus is a family of such pairs (d, p) i.e., a modular signature. An equation will then consist of two derived operations with the same arity, in the following sense.

Definition 7.12. Given any modular signatures S and S', an **S-derived operation of arity S'** is a functor $L\colon S\text{-alg} \to S'\text{-alg}$ over $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$, i.e., making the following triangle commute.



We call operations of S **basic**, by contrast with the **derived** operations of S'.

More concretely, we may introduce derived operations in two stages, as follows:

- an **S-module morphism** $M \to N$ between parametric modules M and N is a natural family of morphisms $(\alpha_T : M(T) \longrightarrow N(T))_{T \in S\text{-alg}}$, such that α_T is a T-module morphism, for each S-algebra T;
- letting $S' = (V_j, q_j)_{j \in J}$, an S-derived operation L of arity S' associates to each S-algebra T an S-module morphism $V_j(T) \to T_{q_j}$, for all $j \in J$.

Equivalently, by adjunction (7.1), an S-derived operation of arity S' associates to each T a single, $\mathbf{Set}^{\mathbb{P}}$-valued T-module morphism $\mathbf{H}_{S'}(T) \to T$, where $\mathbf{H}_{S'}(T) := \sum_{j \in J} V_j(T) \cdot \mathbf{y}_{q_j}$.

Example 7.13. Consider associativity of parallel composition in π-calculus, $P|(Q|R) \equiv (P|Q)|R$: the metavariables are P, Q, and R. The corresponding input arity is $\Theta_{\mathbf{p}}^3$, and the output arity is $\mathbf{p}$. Recalling §4.3, and anticipating on §11.3 below, the basic signature S contains in particular an operation $par : \Theta_{\mathbf{p}}^2 \to \Theta_{\mathbf{p}}$ for parallel composition, and the derived operations for associativity respectively map any algebra $(T, par_T : T_{\mathbf{p}}^2 \to T_{\mathbf{p}}, \ldots)$ to

$$T_{\mathbf{p}}^3 \xrightarrow{par \times T_{\mathbf{p}}} T_{\mathbf{p}}^2 \xrightarrow{par} T_{\mathbf{p}} \qquad \text{and} \qquad T_{\mathbf{p}}^3 \xrightarrow{T_{\mathbf{p}} \times par} T_{\mathbf{p}}^2 \xrightarrow{par} T_{\mathbf{p}}.$$

Returning to the general case, we now define our notion of signature for monads.

Definition 7.14. An **equational modular signature** consists of

- a modular signature S called the modular signature for **operations**, or the **operations** modular signature,
- a modular signature S' called the modular signature for **equations**, or the **equations** modular signature, together with
- a pair of S-derived operations of arity S'.

For any equational modular signature $E = (S, S', L, R)$, an **E-algebra** is an S-algebra T such that the S'-algebra structures $L(T)$ and $R(T)$ coincide, i.e., $L(T) = R(T)$. A morphism of E-algebras is a morphism of S-algebras. We let E-alg denote the category of E-algebras and morphisms between them.

Let us at last define our register.

Definition 7.15. We define the monadic register $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$ for $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$ as follows, for any set $\mathbb{P}$.

Signatures: A signature is an equational modular signature.

Semantics: The abstract signature associated to any equational modular signature E is the forgetful functor E-alg $\to \mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$.

Validity proof: This is Corollary 10.24*(i)* below. □

Let us conclude this subsection by introducing some convenient notation for specifying equations.

Notation 7.16 (Format for equations)**.** *We write any equational modular signature whose equations modular signature is a singleton $S' = (V, p)$, say with derived operations given by*

$$\begin{array}{rcl} V & \to & \Theta_p^2 \\ x & \mapsto & (L, R), \end{array}$$

as

$$x : V \vdash L \equiv R : \Theta_p$$

(leaving the operations modular signature implicit), or even just $L \equiv R$ when the rest may be inferred.

Furthermore, given any common (implicit) operations modular signature S, any family $(x : V_i \vdash L_i \equiv R_i : \Theta_{p_i})_{i \in I}$ will accordingly denote the equational modular signature

- *whose equations modular signature is $S' = (V_i, p_i)_{i \in I}$, and*
- *whose S-derived operations of arity S' are given at any S-algebra X by the morphisms $L_i(X) : V_i(X) \to X_{p_i}$ and $R_i(X) : V_i(X) \to X_{p_i}$, for each $i \in I$.*

Example 7.17. We write associativity from Example 7.13 as just

$$par(P, par(Q, R)) \equiv par(par(P, Q), R).$$

In this case, the argument x is the triple (P, Q, R).

7.4. **Explicit description of initial algebras.** In this final subsection, we provide an explicit description of the initial E-algebra, for any equational modular signature E. We first deal with the case without equations, recalling the standard identification of the initial S-algebra as a free algebra for a suitable endofunctor. In the presence of equations, we then characterise the initial E-algebra as a coequaliser of free algebras.

In order to compute the initial algebra for a modular signature S, we first observe that the induced homogeneous parametric module H_S in fact comes from an endofunctor.

Definition 7.18. For any modular signature S, we define its **associated endofunctor** Σ_S on $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$ as follows.

- For any elementary $d = (F \circ \Theta)^{(p_1, \dots, p_n)}$ and $r \in \mathbb{P}$, let
$$\Sigma_{(d,r)}(P) = (F \circ P)^{(p_1, \dots, p_n)} \cdot \mathbf{y}_r.$$
- For a family $S = (d_i, r_i)$, let $\Sigma_S = \sum_{i \in I} \Sigma_{(d_i, r_i)}$.

Explicitly, letting $\mathbf{C} = [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$, we have for any (d, r):

$$\Sigma_{(d,r)}(P)(X) := (F(P(X + \textstyle\sum_i \mathbf{y}_{p_i}))) \cdot \mathbf{y}_r.$$

In perhaps more elementary terms, we have

$$\begin{array}{rcll} \Sigma_{(d,r)}(P)(X)(r) & = & F(P(X + \sum_i \mathbf{y}_{p_i})) & \\ \Sigma_{(d,r)}(P)(X)(r') & = & \emptyset & \text{for } r' \neq r. \end{array}$$

By construction, we have:

Proposition 7.19. *For any modular signature S, the following square commute,*

$$\begin{array}{ccc} \mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}}) & \xrightarrow{H_S} & \mathbf{Mod}(\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}) \\ \downarrow & & \downarrow \\ [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f & \xrightarrow[\Sigma_S]{} & [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f \end{array}$$

where the right-hand functor maps any pair (T, M) to M.

Let us now turn to the explicit description of the initial S-algebra. The mathematical contents essentially date back to [29].

Proposition 7.20. *For any modular signature S, the forgetful functor S-alg $\to \mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$ is monadic, and furthermore the free Σ_S-algebra $\Sigma_S^*(\mathrm{id})$ on the identity has a canonical S-algebra structure, which is initial.*

Proof. This is Corollary 10.21 below. □

We now seek an explicit description of the initial E-algebra, for any equational modular signature E.

Definition 7.21. Let $E = (S, S', L, R)$ denote any equational modular signature, with $S = (V_i, p_i)_{i \in I}$ and $S' = (W_j, q_j)_{j \in J}$.

- Let $S + S'$ denote the "disjoint union", i.e., the modular signature $(U_k, r_k)_{k \in K}$, where
  - $K = I + J$,
  - (U_k, r_k) is
    - (V_i, p_i) if $k = in_1(i)$ and
    - (W_j, q_j) if $k = in_2(j)$.
- The S'-algebra structures given by $L(spec(S))$ and $R(spec(S))$ on $spec(S)$, together with its canonical S-algebra structure, yield two $(S + S')$-algebra structures. By initiality of $spec(S + S')$, we thus obtain two $(S + S')$-algebra morphisms

$$\check{L}, \check{R} \colon spec(S + S') \to spec(S), \tag{7.2}$$

or equivalently, by Proposition 7.20,

$$\check{L}, \check{R} \colon (\Sigma_S + \Sigma_{S'})^*(\mathrm{id}) \to \Sigma_S^*(\mathrm{id}). \tag{7.3}$$

Theorem 7.22. *For any equational modular signature $E = (S, S', L, R)$, the coequaliser of the pair $\check{L}, \check{R} \colon (\Sigma_S + \Sigma_{S'})^*(\mathrm{id}) \to \Sigma_S^*(\mathrm{id})$ in $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$ admits a unique S-algebra structure such that the coequalising morphism is an S-algebra morphism. Furthermore, this structure is in fact an E-algebra structure. Finally, it makes the coequaliser into an initial E-algebra.*

Remark 7.23.

- The coequaliser may be computed as a pointwise quotient of $\Sigma_S^*(\mathrm{id})$, as follows. Hamana [38] explains how $\Sigma_S^*(\mathrm{id})$ may be defined as a term language, each $\Sigma_S^*(\mathrm{id})(X)(p)$ being the set of terms of type p, with free variables of each type $q \in \mathbb{P}$ in $X(q)$ — the pair (X, p) may be thought of as a sequent $X \vdash p$. Now, coequalisers (as all limits and colimits) are pointwise in functor categories [52, §V.4], so for any $X \in \mathbf{Set}^{\mathbb{P}}$ and $p \in \mathbb{P}$, $spec(S)(X)(p)$ is the coequaliser of

$$(\Sigma_S + \Sigma_{S'})^*(\mathrm{id})(X)(p) \to \Sigma_S^*(\mathrm{id})(X)(p)$$

in $\mathbf{Set}$. And this is well known to be the quotient of $\Sigma_S^*(\mathrm{id})(X)(p)$ by the smallest equivalence relation identifying all $\check{L}_{X,p}(e)$ with $\check{R}_{X,p}(e)$, for all $e \in (\Sigma_S + \Sigma_{S'})^*(\mathrm{id})(X)(p)$. Intuitively, following Hamana again, $(\Sigma_S + \Sigma_{S'})^*(\mathrm{id})$ is an extension of the term language $\Sigma_S^*(\mathrm{id})$ with operations, say o_j, with arities (V_j, q_j), for all j (where $S' = (V_j, q_j)_{j \in J}$ as before), and $\check{L}$ and $\check{R}$ inductively translate this language to $\Sigma_S^*(\mathrm{id})$ by interpreting o_j using $L_j(\Sigma_S^*(\mathrm{id}))$ and $R_j(\Sigma_S^*(\mathrm{id}))$, respectively.
- We could consider computing the coequaliser of a simpler parallel pair

$$spec(S') \to spec(S), \tag{7.4}$$

constructed similarly. Let us show on a simple example that this does not compute the desired functor. The intuition is that this coequaliser identifies terms modulo a relation which is not a congruence.

Let $\mathbb{P} = 1$, and S consist of a single, binary operation, i.e., $S = \{(\Theta^2, \star)\}$. Thus, an S-algebra is merely a monad on sets, equipped with a binary T-module morphism

$b\colon T^2 \to T$. Furthermore, let S' consist of a single, ternary operation t. Finally, for any S-algebra (T, b), let $L(T)$ and $R(T)$ denote the T-module morphisms

$$T^3 \xrightarrow{b\times T} T^2 \xrightarrow{b} T \qquad\qquad T^3 \xrightarrow{T\times b} T^2 \xrightarrow{b} T.$$

By Proposition 7.20, for any set X, $spec(S)(X)$ consists of binary trees with leaves in X. Similarly, $spec(S')(X)$ consists of ternary trees with leaves in X. Now, the parallel pair (7.4) maps ternary trees to binary trees, replacing any ternary node (t_1, t_2, t_3), respectively with $((t_1, t_2), t_3)$ and $(t_1, (t_2, t_3))$. Thus, e.g., assuming $x \in X$, the binary trees $(((x, x), x), x)$ and $((x, (x, x)), x)$, having an even number of leaves, are not in the image of the parallel pair, hence are not identified in the coequaliser.

- A perhaps higher-level understanding of this, which will be developped in §10, is as follows. The derived operations induce monad morphisms $(S')^\star \to S^\star$, of which the desired monad $E^\star$ is the coequaliser in $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$. However, coequalisers of monads are generally not pointwise [2]. Fortunately, **reflexive** coequalisers are, so the desired monad may be computed as the pointwise coequaliser of the obvious parallel pair $S^\star + (S')^\star \to S^\star$. Finally, roughly because $(-)^\star$ is a left adjoint in this case, we have $S^\star + (S')^\star \cong (S + S')^\star$, which directly leads to our formula.

8. Registers for (state) functors

In this section, we define a register $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$, which is a variant of $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$ for the case of state functors. Operations, equations, and models will be defined exactly as for monads, and a signature in $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ will again consist of families of operations and equations, the only difference being that instead of parametric modules, we will use **facets**. We start by presenting the auxiliary notion of facets, which is a simple variant of the parametric modules of §7. Next, we introduce a register $\mathbf{Reg}^0[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ for operations, and then a refinement $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ with equations. Again, most validity proofs are deferred to §10.

8.1. **Facets.** We start in this section by introducing facets. Let us first explain how they naturally come up in the case of call-by-value, simply-typed λ-calculus. We have seen in §4.1 that the source state functor S_1 for call-by-value, simply-typed λ-calculus consists of application binary trees. The goal now is to design a register for specifying such a functor S_1. Intuitively, it has two (type-indexed families of) operations:

- a first operation for injecting values into application binary trees, of type $X_A \to S_1(X)_A$ for all $X \in \mathbf{Set}^{\mathbb{P}}$, and
- a second operation for application, of type $S_1(X)_{A\to B} \times S_1(X)_A \to S_1(X)_B$, for all $X \in \mathbf{Set}^{\mathbb{P}}$.

The type for application is really similar to what we had in §7: for any type $A \in \mathbb{P}$, denoting by $\Theta_A\colon [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}] \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}]$ the functor defined by

$$\Theta_A(S)(X) = S(X)_A,$$

we have an arity

$$\Theta_{A\to B} \times \Theta_A \to \Theta_B.$$

The type for value injection does not make any sense in the context of modules over monads, because the functor $M(T)(X) = X_A$ does not form a module. But here in the context of state functors, we may well define $\mathbf{I}_A \colon [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}] \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}]$ by

$$\mathbf{I}_A(S)(X) = X_A,$$

for any $A \in \mathbb{P}$.

Such functors Θ_A and $\mathbf{I}_A$ are examples of facets, which we now introduce more formally.

Definition 8.1. For any categories $\mathbf{C}$ and $\mathbf{D}$, a **facet** for $[\mathbf{C}, \mathbf{D}]_f$ is a finitary functor $[\mathbf{C}, \mathbf{D}]_f \to [\mathbf{C}, \mathbf{Set}]_f$.

Let $\mathbf{Facet}(\mathbf{C}, \mathbf{D}) \coloneqq [[\mathbf{C}, \mathbf{D}]_f, [\mathbf{C}, \mathbf{Set}]_f]_f$ denote the category of facets.

Notation 8.2. *We abbreviate* $\mathbf{Facet}(\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}})$ *to* $\mathbf{Facet}(\mathbb{P}, \mathbb{S})$*, for readability.*

Definition 8.3. Here are a few basic constructions of facets, the first three in the general case, and the next three for $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$.

- Any functor $F \colon \mathbf{D} \to \mathbf{Set}$ induces a facet Φ_F defined by $\Phi_F(P)(C) = F(P)(C)$.
- Similarly, any functor $G \colon \mathbf{C} \to \mathbf{Set}$ induces a facet Ψ_G defined by $\Psi_G(P)(C) = G(C)$.
- For any facets F and G, the product $F \times G$ in the (functor) category of facets is again a facet.
- For any $p \in \mathbb{P}$, let $\Theta_p = \Phi_{\mathrm{ev}_p}$.
- For any $s \in \mathbb{S}$, let $\mathbf{I}_s = \Psi_{\mathrm{ev}_s}$.
- For any facet F for $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]$ and $p_1, \ldots, p_n \in \mathbb{P}$, let

$$F^{(p_1,\ldots,p_n)}(P)(X) = F(P(X + \mathbf{y}_{p_1} + \ldots + \mathbf{y}_{p_n})).$$

Remark 8.4. Let Θ denote the identity endofunctor on $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$, and

$$\mathbf{I} \colon [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$$

the constant functor mapping anything to the identity endofunctor. Post-composing with evaluation at any $p \in \mathbb{P}$, resp. $s \in \mathbb{S}$, we recover the facets $\mathbf{I}_p$ and Θ_s.

Remark 8.5. The notation Θ introduced above for the identity endofunctor is compatible with the one denoting the parametric module mapping a monad T on $\mathbf{Set}^{\mathbb{P}}$ to T as a module over itself (Example 7.4) in the sense that, for example, the functor underlying the module T coincides with the functor underlying the monad T.

Example 8.6. The arities for application binary trees will be given by $\mathbf{I}_A \to \Theta_A$ and $\Theta_{A \to B} \times \Theta_A \to \Theta_B$, for all types A and B.

8.2. **The register $\mathbf{Reg}^0[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ for specifying operations.** In order to adapt the notion of signature for operations from $\mathbf{RegMnd}^0_f(\mathbf{Set}^{\mathbb{P}})$, we merely need to adapt the notion of elementariness, which becomes the following:

Definition 8.7. A facet for $[\mathbf{Set}^{\mathbb{P}}, \mathbf{S}et^{\mathbb{S}}]_f$ is **elementary** if it is isomorphic to some finite product of facets of the shape $(H \circ \langle \mathbf{I}, \Theta \rangle)^{(p_1,\ldots,p_n)}$ for some $p_1, \ldots, p_n \in \mathbb{P}$ and finitary functor $H \colon \mathbf{Set}^{\mathbb{P}} \times \mathbf{Set}^{\mathbb{S}} \to \mathbf{Set}$.

Example 8.8. Typically, any product of facets of the shape $\mathbf{I}_p^{(p_1,\ldots,p_n)}$ or $\Theta_s^{(p_1,\ldots,p_n)}$, for some $p, p_1, \ldots, p_n \in \mathbb{P}$ and $s \in \mathbb{S}$, is elementary.

Definition 8.9. A **facet-based signature** is a family of pairs (d, s) consisting of an elementary facet d and a transition type $s \in \mathbb{S}$.

Definition 8.10. For any facet-based signature $S = (d_i, s_i)_{i \in I}$, an **S-algebra** is a finitary functor $F \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, equipped with natural transformations $d_i(F) \to \Theta_{s_i}(F)$ for all $i \in I$. A **morphism of S-algebras** is a natural transformation commuting with these morphisms. Let S-alg denote the category of S-algebras, and $U_S \colon S\text{-alg} \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ the forgetful functor.

Definition 8.11. For any sets $\mathbb{P}$ and $\mathbb{S}$, we define the monadic register $\mathbf{Reg}^0[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ as follows.

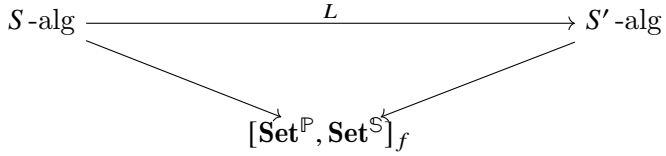
Signatures: A signature is a facet-based signature.

Semantics: The abstract signature associated to any facet-based signature S is the forgetful functor $S\text{-alg} \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$.

Validity proof: By Proposition 8.19 below. □

8.3. The register $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$.

Definition 8.12. Given any facet-based signatures S and S', an **S-derived operation of arity S'** is a functor $L \colon S\text{-alg} \to S'\text{-alg}$ over $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$, i.e., making the following triangle commute.



We call **basic** the operations of S, by contrast with the **derived** operations of S'.

More concretely, as in Definition 7.12, we may introduce derived operations in two stages, as follows:

- an **S-facet morphism** $M \to N$ between facets M and N is a natural family of natural transformations $(\alpha_F \colon M(F) \longrightarrow N(F))_{F \in S\text{-alg}}$;
- letting $S' = (V_j, q_j)_{j \in J}$, an S-derived operation L of arity S' associates to each S-algebra F an S-facet morphism $V_j \to \Theta_{q_j}$, for all $j \in J$.

Definition 8.13. An **equational facet-based signature** consists of

- a facet-based signature S called the **operations** facet-based signature,
- a facet-based signature S' called the **equations** facet-based signature, and
- a pair of S-derived operations of arity S'.

For any equational facet-based signature $E = (S, S', L, R)$, an **E-algebra** is an S-algebra F such that the S'-algebra structures $L(F)$ and $R(F)$ coincide, i.e., $L(F) = R(F)$. A morphism of E-algebras is a morphism of S-algebras. We let E-alg denote the category of E-algebras and morphisms between them.

Definition 8.14. For any sets $\mathbb{P}$ and $\mathbb{S}$, we define the monadic register $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ for $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ as follows.

Signatures: A signature is an equational facet-based signature.

Semantics: The abstract signature associated to any equational facet-based signature E is the forgetful functor $E\text{-alg} \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$.

Validity proof: By Corollary 10.28 below. □

Remark 8.15. Any finitary functor F admits a trivial signature consisting of the family $((F_s \circ \mathbf{I}) \to \Theta_s)_{s\in\mathbb{S}}$ of operations, which does not prevent other signatures from being more convenient (see the case of Example 8.6). Here are a few examples from §4:

Language	State functor	Trivial signature
$\overline{\lambda}\mu$	$S_1(X) = S_2(X)$ $= (X_\mathbf{p} \times X_\mathbf{s}, X_\mathbf{p}, X_\mathbf{s})$	$\langle -\vert -\rangle\colon \mathbf{I}_\mathbf{p} \times \mathbf{I}_\mathbf{s} \to \Theta_\mathbf{c}$ $\eta_\mathbf{p}\colon \mathbf{I}_\mathbf{p} \to \Theta_\mathbf{p}$ $\eta_\mathbf{s}\colon \mathbf{I}_\mathbf{s} \to \Theta_\mathbf{s}$
π	$S_1(X) = S_2(X) = X_\mathbf{p}$	$\mathbf{I}_\mathbf{p} \to \Theta$
Call-by-value, simply-typed λ	$S_2(X) = X$	$\eta_t : \mathbf{I}_t \to \Theta_t$ (for all t)
Positive GSOS specifications	$S_1(X) = X$ $S_2(X) = \mathbb{A} \times X$	$\mathbf{I} \to \Theta$ $\mathbb{A} \times \mathbf{I} \to \Theta$

Notation 8.16. *In examples, we will present equational facet-based signatures as families of equations. For this, we will use Notation 7.16, which extends to facet-based equations. E.g., Example 7.17 applies verbatim for associativity of multiset union in the target state functor for differential λ-calculus.*

8.4. **Explicit description of initial algebras.** In this section, we state monadicity of the register $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ of equational facet-based signatures, and give our explicit description of initial algebras. We first deal with facet-based signatures S, identifying the initial S-algebra as a free algebra for a suitable endofunctor. We then characterise the initial E-algebra as a coequaliser of free algebras, for any equational facet-based signature E.

Definition 8.17. For any facet-based signature S, we define its **associated endofunctor** Σ_S on $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ as follows.

- For any elementary $d = (F \circ \langle \mathbf{I}, \Theta\rangle)^{(p_1,\dots,p_n)}$ and $s \in \mathbb{S}$, let
$$\Sigma_{(d,s)}(P) = (F \circ \langle \mathrm{id}, P\rangle)^{(p_1,\dots,p_n)} \cdot \mathbf{y}_s.$$
- For a family $S = (d_i, s_i)$, let $\Sigma_S := \sum_{i\in I} \Sigma_{(d_i,s_i)}$.

Remark 8.18. Explicitly, we have for any (d, s):
$$\Sigma_{(d,s)}(P)(X) := (F(X + \textstyle\sum_i \mathbf{y}_{p_i}, P(X + \sum_i \mathbf{y}_{p_i}))) \cdot \mathbf{y}_s.$$

Proposition 8.19. *For any facet-based signature S, the forgetful functor $U_S\colon S\text{-alg} \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ is monadic. Furthermore, the free S-algebra on a functor $X \in [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ is the free Σ_S-algebra $\Sigma_S^*(X)$, as characterised in Proposition 5.9.*

Proof. A direct consequence of Proposition 10.25 below. □

We now want to characterise the initial E-algebra, for any equational facet-based signature E. The development closely follows §7.4.

Definition 8.20. Let $E = (S, S', L, R)$ denote any equational facet-based signature, with $S = (V_i, p_i)_{i\in I}$ and $S' = (W_j, q_j)_{j\in J}$.

- Let $S + S'$ denote the "disjoint union", i.e., the facet-based signature $(U_k, r_k)_{k\in K}$, where
  - $K = I + J$,
  - (U_k, r_k) is

  - (V_i, p_i) if $k = in_1(i)$ and
  - (W_j, q_j) if $k = in_2(j)$.
- The S'-algebra structures given by $L(spec(S))$ and $R(spec(S))$ on $spec(S)$, together with its canonical S-algebra structure, yield two $(S + S')$-algebra structures. By initiality of $spec(S + S')$, we thus obtain two $(S + S')$-algebra morphisms

$$\check{L}, \check{R} \colon spec(S + S') \to spec(S). \tag{8.1}$$

Theorem 8.21. *For any equational facet-based signature $E = (S, S', L, R)$, the coequaliser of the pair $\check{L}, \check{R} \colon spec(S+S') \to spec(S)$ in $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ admits a unique S-algebra structure such that the coequalising morphism is an S-algebra morphism. Furthermore, this structure is in fact an E-algebra structure. Finally, it makes the coequaliser into an initial E-algebra.*

9. General registers

We have now introduced all the needed registers for our register $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$ of Definition 5.15 to make sense, including two registers featuring equations, respectively for monads (Definition 7.15) and functors (Definition 8.14). In this section, in preparation for the missing validity proofs and the announced explicit descriptions of initial algebras, we introduce a fundamental register featuring equations, for a general category, $\mathbf{ES_C}$, whose signatures are Fiore and Hur's equational systems [28]. In order to deal more specifically with monads, we then introduce a second register, $\mathbf{PSEF_C}$, based on Fiore, Plotkin, and Turi's pointed strong endofunctors, which incorporates variable binding and substitution. We then "merge" both registers into a single register $\mathbf{MES_C}$, suited for syntax with variable binding, substitution, and equations.

9.1. **Equational systems.** A general device for constructing categories of algebras is Fiore and Hur's **equational systems** [28]. In this section, we view equational systems on a category $\mathbf{C}$ as signatures of a register $\mathbf{ES_C}$ for $\mathbf{C}$, called the **equational register**, which refines with equations the endofunctorial register $\mathbf{EF_C}$ (see Definition 5.10).

Briefly, an equational system consists of two parts:

- an endofunctor Σ on $\mathbf{C}$, which intuitively specifies operations,
- and equations.

We present them in a slightly non-standard way, and make the link with the original in Propositions 9.5 and 9.8 below.

Before stating the definition, let us briefly introduce the relevant notion of equation in the case of endofunctors on sets. Standardly, given a finitary endofunctor Σ on $\mathbf{Set}$ an equation is merely a pair of terms, i.e., an element of the product $\Sigma^*(n)^2$, if the considered terms have n variables (recalling that Σ^* denotes the free monad on any finitary endofunctor Σ). E.g., for associativity of a binary operation we would have $n = 3$. In order to deal with variable binding, let us observe that by Yoneda a term in $\Sigma^*(n)$ is equivalent to a map $n \to \Sigma^*$. Generalising from n to an arbitrary endofunctor, we put:

Definition 9.1. For any endofunctor Γ and monad T on a category $\mathbf{C}$, a **functorial T-term** of arity Γ is a natural transformation $\Gamma \to T$.

Example 9.2. Let us consider the finitary endofunctor Σ on sets defined by $\Sigma(X) = X^2$, so that $\Sigma^*(X)$ denotes the set of binary trees with leaves in X, as generated by the following grammar,

$$e, f ::= x \mid \mathrm{op}(e, f)$$

where x ranges over X. Since Σ-algebras are sets equipped with a binary operation, a natural equation to impose on them is associativity. Taking $\Gamma(X) = X^3$, the relevant functorial Σ^*-terms L and R of arity Γ are defined by $L(x_1, x_2, x_3) = \mathrm{op}(\mathrm{op}(x_1, x_2), x_3)$, and $R(x_1, x_2, x_3) = \mathrm{op}(x_1, \mathrm{op}(x_2, x_3))$.

Example 9.3. We will do this right in Example 9.26 below, but for illustrative purposes, let us describe a failed attempt at specifying pure λ-terms modulo β. We first take $\mathbf{C} = [\mathbf{Set}, \mathbf{Set}]_f$ to consist of finitary endofunctors on sets, and $\Sigma(X)(n) = X(n)^2 + X(n+1)$, the first summand modelling application and the second abstraction. Indeed, any algebra X comes equipped with maps $app_n \colon X(n)^2 \to X(n)$ and $\lambda_n \colon X(n+1) \to X(n)$, for all n. The first member of the β-equation, $\mathrm{app}_n(\lambda_n(e), f)$, would be modelled by setting $\Gamma(X)(n) = X(n+1) \times X(n)$ and taking the natural transformation $\Gamma \to \Sigma^*$ mapping any $(e, f) \in X(n+1) \times X(n)$ to $\mathrm{app}_n(\lambda_n(e), f)$ (omitting the monad unit $\eta \colon \mathrm{id} \to \Sigma^*$). This works fine, but we have neglected to build substitution into the model, so we cannot define the right-hand side of β. This will be rectified in Example 9.26.

The following should now look natural.

Definition 9.4. An **equational system** $\mathbb{E} = (\mathbf{C} \colon \Gamma \vdash L = R : \Sigma)$ consists of

- a locally finitely presentable category $\mathbf{C}$,
- finitary endofunctors Σ and Γ, together with
- functorial Σ^*-terms L and R of arity Γ.

This differs slightly from Fiore and Hur's [28] definition, so let us readily bridge the gap.

Proposition 9.5. *Given any locally finitely presentable category $\mathbf{C}$, there are natural isomorphisms (in Σ and Γ) between*

(i) *functorial Σ^*-terms of arity Γ, i.e., natural transformations $\Gamma \to \Sigma^*$,*
(ii) *monad morphisms $\Gamma^* \to \Sigma^*$,*
(iii) *functors Σ-alg $\to \Gamma$-alg over $\mathbf{C}$,*
(iv) *monad morphisms $(\Sigma+\Gamma)^* \to \Sigma^*$ with section the canonical morphism $\Sigma^* \to (\Sigma+\Gamma)^*$,*
(v) *functors Σ-alg $\to (\Sigma+\Gamma)$-alg over $\mathbf{C}$ which are sections of the canonical functor $(\Sigma+\Gamma)$-alg $\to \Sigma$-alg, and*
(vi) *natural transformations $\Gamma \circ U_\Sigma \to U_\Sigma$.*

Proof.

- *(i) ⇔ (ii)* This is precisely the universal property of Γ^*.
- *(ii) ⇔ (iii)* This is precisely Corollary 2.36.
- *(ii) ⇔ (iv)* By $(\Sigma+\Gamma)^* \cong \Sigma^* + \Gamma^*$, which holds (in $\mathbf{Mnd}_f(\mathbf{C})$) because $(-)^*$ is a left adjoint, hence preserves coproducts.
- *(iii) ⇔ (v)* By $(\Sigma+\Gamma)$-alg $\cong \Sigma$-alg $\times_{\mathbf{C}} \Gamma$-alg, with the canonical functor $(\Sigma+\Gamma)$-alg $\to \Sigma$-alg as left projection.
- *(iii) ⇔ (vi)* By merely unfolding definitions. □

Remark 9.6. The expert will have noticed that Fiore and Hur's equational systems use *(iii)*, in a slightly more general setting: they do not assume $\mathbf{C}$ to be locally finitely presentable, nor Σ and Γ to be finitary.

Let us now define the models of an equational system.

Definition 9.7. For any equational system $\mathbb{E} = (\mathbf{C} \colon \Gamma \vdash L = R : \Sigma)$, an **$\mathbb{E}$-algebra** is a Σ-algebra X whose induced Σ^*-algebra structure coequalises

$$L_X, R_X \colon \Gamma(X) \rightrightarrows \Sigma^*(X).$$

Let $\mathbb{E}$-alg denote the category of $\mathbb{E}$-algebras, with Σ-algebra morphisms between them, and let $U_{\mathbb{E}} \colon \mathbb{E}\text{-alg} \to \mathbf{C}$ denote the forgetful functor.

Let us readily transfer this definition across the various correspondences of Proposition 9.5.

Proposition 9.8. *For any equational system $\mathbb{E} = (\mathbf{C} \colon \Gamma \vdash L = R : \Sigma)$ and Σ^*-algebra $a \colon \Sigma^*(X) \to X$, the following are equivalent:*

(a) *a coequalises $L_X, R_X \colon \Gamma(X) \to \Sigma^*(X)$,*
(b) *a coequalises the corresponding morphisms $\Gamma^*(X) \to \Sigma^*(X)$,*
(c) *the induced Σ-algebra structure $\Sigma(X) \to X$ belongs to the equaliser of the corresponding functors $\Sigma\text{-alg} \to \Gamma\text{-alg}$ over $\mathbf{C}$,*
(d) *a coequalises the corresponding morphisms $(\Sigma + \Gamma)^*(X) \to \Sigma^*(X)$,*
(e) *the induced Σ-algebra structure $\Sigma(X) \to X$ belongs to the equaliser of the corresponding functors $\Sigma\text{-alg} \to (\Sigma + \Gamma)\text{-alg}$ over $\mathbf{C}$,*
(f) *the corresponding natural transformations $\Gamma \circ U_\Sigma \to U_\Sigma$ have the same components at the induced Σ-algebra structure $\Sigma(X) \to X$.*

Proof.

- *(a) $\Leftrightarrow$ (c)* The corresponding functors $\Sigma\text{-alg} \to \Gamma\text{-alg}$ map the induced Σ-algebra to $\Gamma(X) \xrightarrow{K} \Sigma^*(X) \xrightarrow{a} X$, for $K = L, R$, so both sides unfold to the same thing.
- *(b) $\Rightarrow$ (a)* Follows from the fact that precomposing the induced morphisms $\Gamma^* \to \Sigma^*$ by $\Gamma \to \Gamma^*$ yields the original L and R by construction.
- *(c) $\Rightarrow$ (b)* This holds because *(b)* is equivalent to equality of induced Γ^*-algebra structures, which is further equivalent to equality of induced Γ-algebra structures.

The rest follows easily. □

Remark 9.9. The equivalence of *(a)* and *(c)* entails that our notion of algebra coincides with Fiore and Hur's (apart from the difference noted in Remark 9.6).

Remark 9.10. Families $(t_i = u_i)_i$ of equations are covered by taking the coproduct of all involved endofunctors, say Γ_i, and the pointwise copairing of functorial terms.

Definition 9.11. For a given locally finitely presentable category $\mathbf{C}$, we define the monadic register $\mathbf{ES}_\mathbf{C}$, called the **equational register**, as follows.

Signatures: A signature is just an equational system.
Semantics: The abstract signature associated to a signature $\mathbb{E}$ is the forgetful functor $\mathbb{E}\text{-alg} \to \mathbf{C}$.

Validity proof: By Theorem 10.14 below. □

Example 9.12. Given a cocomplete category $\mathbf{C}$, the endofunctor register $\mathbf{EF}_{\mathbf{C}}$ is a subregister of $\mathbf{ES}_{\mathbf{C}}$, by mapping any finitary endofunctor Σ to the (finitary) equational system given by taking $\Gamma = \emptyset$, and the unique L and R.

9.2. **The register $\mathbf{PSEF}_{\mathbf{C}}$ for monoids.** In the previous subsection, we reviewed the fundamental register available for pretty general categories. Here we review the fundamental register available for the category of monoids in a convenient monoidal category, essentially due to Fiore, Plotkin, and Turi [29].

Signatures in the register $\mathbf{PSEF}_{\mathbf{C}}$ will be pointed strong endofunctors on the monoidal category $\mathbf{C}$. Indeed the crucial virtue of such a pointed strong endofunctor Σ is that, beside the (monadic) abstract signature on $\mathbf{C}$ induced by the underlying endofunctor, it yields a (monadic) abstract signature $U_\Sigma \colon \Sigma\text{-}\mathbf{Mon} \to \mathbf{Mon}(\mathbf{C})$ on $\mathbf{Mon}(\mathbf{C})$, the point being that in concrete cases monoids in $\mathbf{C}$ are intuitively objects equipped with substitution. Let us thus describe the announced monadic abstract signature. We start by recalling what is a pointed strong endofunctor, and showing how it yields a parametric module (in the sense of Definition 2.13) on $\mathbf{C}$.

Pointed strong endofunctors

Definition 9.13. A **pointed strong endofunctor** on a monoidal category $(\mathbf{C}, \otimes, I)$ is an endofunctor Σ equipped with a **pointed strength**, i.e., a natural transformation with components

$$st_{X,Y} : \Sigma(X) \otimes Y \to \Sigma(X \otimes Y)$$

between functors $\mathbf{C} \times I/\mathbf{C} \to \mathbf{C}$, making the following diagrams commute (pointed objects $e_Y : I \to Y$ are denoted by their codomains Y for readability),

$$\begin{array}{ccccc}
(\Sigma(X) \otimes Y) \otimes Z & & \xrightarrow{\quad\alpha_{\Sigma(X),Y,Z}\quad} & & \Sigma(X) \otimes (Y \otimes Z) \\
\downarrow{\scriptstyle st_{X,Y} \otimes Z} & & & & \downarrow{\scriptstyle st_{X,Y \dot\otimes Z}} \\
\Sigma(X \otimes Y) \otimes Z & \xrightarrow[st_{X\otimes Y,Z}]{} & \Sigma((X \otimes Y) \otimes Z) & \xrightarrow[\Sigma(\alpha_{X,Y,Z})]{} & \Sigma(X \otimes (Y \otimes Z))
\end{array}$$

$$\begin{array}{ccc}
\Sigma(X) & \xrightarrow{\quad\rho_{\Sigma(X)}\quad} & \Sigma(X) \otimes I \\
& {\scriptstyle \Sigma(\rho_X)} \searrow \quad \swarrow {\scriptstyle st_{X,I}} & \\
& \Sigma(X \otimes I) &
\end{array}$$

where $I/\mathbf{C}$ inherits monoidal structure $(\dot I, \dot\otimes, \ldots)$ from $\mathbf{C}$ in the obvious way [62].

Remark 9.14. There is a simpler notion of **strong** endofunctor, which also involves a natural transformation $st_{X,Y} : \Sigma(X) \otimes Y \to \Sigma(X \otimes Y)$, although with Y not pointed. Strong endofunctors embed into pointed strong endofunctors, which are thus more general. This generalisation is necessary to cover variable binding, as shown by the following example.

Example 9.15. For defining the λ-calculus syntax, we take $\mathbf{C} := [\mathbf{Set}, \mathbf{Set}]_f$ to be the category of finitary endofunctors on sets with the composition tensor product, and $\Sigma(X)(n) = X(n)^2 + X(n+1)$ (using the equivalence $[\mathbf{Set}, \mathbf{Set}]_f \simeq [\mathbf{Set}_f, \mathbf{Set}]$, where $\mathbf{Set}_f$ is the full category spanning finite ordinals). The tensor product is defined on $[\mathbf{Set}_f, \mathbf{Set}]$ the coend formula [52, §IX.6]

$$(X \otimes Y)(n) = \int^m X(m) \times Y(n)^m,$$

which is in one-to-one correspondence with $X(Y(n))$. Elements of $(X \otimes Y)(n)$ may be thought of as explicit substitutions $x[\![\sigma]\!]$, with $x \in X(m)$ and $\sigma : m \to Y(n)$, considered equivalent up to standard equations, compactly summarised by $(f \cdot x)[\![\sigma]\!] = x[\![\sigma \circ f]\!]$, for all $f : m \to p$, $x \in X(m)$ and $y : p \to Y(n)$, where $f \cdot x$ is shorthand for $X(f)(x)$.

A monoid structure on any $X \in \mathbf{C}$ thus amounts to

- a substitution operation $X \otimes X \to X$ mapping any such explicit substitution $x[\![\sigma]\!] \in (X \otimes X)(n)$ to some proper substitution, which we denote by $x[\sigma] \in X(n)$,
- together with a morphism $I \to X$, which, because $I(n) = \mathrm{id}(n) = n$, amounts to identifying available variables within each $X(n)$.

These data are of course required to satisfy the usual associativity and unitality conditions, which amount to standard substitution lemmas.

Returning to pointed strengths $(st_{X,Y})_n : \Sigma(X)(Y(n)) \to \Sigma(X(Y(n)))$, intuitively, they describe the behaviour of substitution w.r.t. application and abstraction. E.g., for application, we define it to map $(in_1(x_1, x_2))[\![\sigma]\!]$ to $in_1(x_1[\![\sigma]\!], x_2[\![\sigma]\!])$. Abstraction is the point where **pointedness** of st comes into play. Indeed, supposing that Y is equipped with a point $e_Y : I \to Y$, we may define $\sigma^{\uparrow} : m + 1 \to Y(n + 1)$ by copairing of

$$m \xrightarrow{\sigma} Y(n) \xrightarrow{Y(in_1)} Y(n+1) \qquad \text{and} \qquad 1 \xrightarrow{(e_Y)_1} Y(1) \xrightarrow{Y(in_2)} Y(n+1).$$

We use this in defining the strength to map any $in_2(x)[\![\sigma]\!]$, where $x \in X(m+1)$ and $\sigma : m \to Y(n)$, to $in_2(x[\![\sigma^{\uparrow}]\!])$.

The parametric module Σ^{stmod} Let us now show how every pointed strong endofunctor induces a parametric module.

Definition 9.16. We define the parametric module Σ^{stmod} on $\mathbf{C}$ as assigning to any monoid $X \in \mathbf{C}$, the object $\Sigma(X)$, equipped with the action given by the composite

$$\Sigma(X) \otimes X \to \Sigma(X \otimes X) \to \Sigma(X).$$

The verification that this assignment indeed defines a parametric module is straightforward.

The category Σ-Mon of models We now introduce Σ-monoids. These are exactly the classical Σ-monoids, which we present from a point of view better suited to our purpose. They are monoids equipped with a "compatible" algebra structure:

Definition 9.17. A **Σ-monoid** is a monoid X, equipped with an X-module morphism $\nu_X : \Sigma^{stmod}(X) \to X$. A Σ-monoid morphism is a monoid morphism $f : X \to Y$ making the following square commute.

$$\begin{array}{ccc} \Sigma^{stmod}(X) & \xrightarrow{\Sigma^{stmod}(f)} & \Sigma^{stmod}(Y) \\ {\scriptstyle \nu_X}\downarrow & & \downarrow{\scriptstyle \nu_Y} \\ X & \xrightarrow[f]{} & Y \end{array}$$

We let Σ-**Mon** denote the category of Σ-monoids and morphisms between them, while $U_\Sigma : \Sigma\text{-}\mathbf{Mon} \to \mathbf{Mon}(\mathbf{C})$ denotes the obvious forgetful functor.

Let us first prove that this agrees with the standard definition.

Proposition 9.18. *The category* Σ-**Mon** *is isomorphic over* $\mathbf{Mon}(\mathbf{C})$ *to the following category.*

- *Objects are monoids X equipped with Σ-algebra structure $\nu_X : \Sigma(X) \to X$ making the diagram*

$$\begin{array}{ccccc} \Sigma(X) \otimes X & \xrightarrow{st_{X,X}} & \Sigma(X \otimes X) & \xrightarrow{\Sigma(m_X)} & \Sigma(X) \\ \downarrow{\scriptstyle \nu_X \otimes X} & & & & \downarrow{\scriptstyle \nu_X} \\ X \otimes X & & \xrightarrow[m_X]{} & & X \end{array} \tag{9.1}$$

  commute.
- *Morphisms are morphisms in $\mathbf{C}$ which are both monoid and Σ-algebra morphisms.*

Proof. By definition of Σ^{stmod}, it is equivalent for a morphism $\Sigma(X) \to X$ to be an X-module morphism, and for the condition (9.1) to hold. □

Let us finally check that we have defined an abstract signature.

Theorem 9.19. *For any finitary, pointed strong endofunctor on a convenient monoidal category $\mathbf{C}$, the category Σ-$\mathbf{Mon}$ and the forgetful functor $U_\Sigma : \Sigma\text{-}\mathbf{Mon} \to \mathbf{Mon}(\mathbf{C})$ form a monadic abstract signature. Equivalently, the forgetful functor $U_\Sigma : \Sigma\text{-}\mathbf{Mon} \to \mathbf{Mon}(\mathbf{C})$ is monadic.*

Proof. We have a commuting triangle

$$\begin{array}{ccc} \Sigma\text{-}\mathbf{Mon} & \xrightarrow{U_\Sigma} & \mathbf{Mon}(\mathbf{C}) \\ & {\scriptstyle \mathbf{U}^{\Sigma\text{-}\mathbf{Mon}}} \searrow \quad \swarrow {\scriptstyle \mathbf{U}^{\mathbf{Mon}(\mathbf{C})}} & \\ & \mathbf{C}, & \end{array}$$

where, by [28, §7.2], $\mathbf{U}^{\Sigma\text{-}\mathbf{Mon}}$ and $\mathbf{U}^{\mathbf{Mon}(\mathbf{C})}$ are both monadic, the corresponding monads being finitary (noting that $\mathbf{Mon}(\mathbf{C}) \cong 0\text{-}\mathbf{Mon}$). By [3, Remark 2.78] with $\lambda = \omega$, it follows that all three categories are locally finitely presentable, hence in particular cocomplete. The result thus follows by Proposition 2.30. □

We now want to recall a standard characterisation of the monad induced by the monadic functor $U^{\Sigma\text{-}\mathbf{Mon}}$, but before that let us fix some notation.

Notation 9.20. *Let us refine Notation 5.7. For a pointed strong endofunctor Σ on a convenient monoidal category $\mathbf{C}$, $\Sigma^\star(0)$ might be read as different objects of $\mathbf{C}$, according to whether Σ is viewed as a finitary endofunctor or a finitary pointed strong endofunctor. Since Σ-monoids are also monadic over monoids, it might even be understood as a monoid in $\mathbf{C}$. We choose the following convention:*

- *$\Sigma^\star$ denotes the "free Σ-monoid" monad $\mathbf{C}$,*
- *$\Sigma^\circledast$ denotes the "free Σ-monoid" monad on $\mathbf{Mon}(\mathbf{C})$,*
- *Σ^* denotes the "free Σ-algebra" monad on $\mathbf{C}$.*

Proposition 9.21. *For any pointed strong endofunctor Σ on a convenient monoidal category $\mathbf{C}$, the monad $\Sigma^\star$ induced by the right adjoint functor $\Sigma\text{-}\mathbf{Mon} \to \mathbf{Mon}(\mathbf{C}) \to \mathbf{C}$ maps any object $C \in \mathbf{C}$ to $\mu A.(I + \Sigma(A) + C \otimes A)$. As a consequence of Proposition 5.9, the carrier $\Sigma^\star(0)$ of the initial Σ-monoid is then isomorphic to $\Sigma^*(I)$.*

Proof. This is more or less known since [29, 28, 30], see [23, Theorem 2.15] for an explicit, complete, yet slightly more general statement. □

Let us finally define our register for monoids.

Definition 9.22. For any convenient monoidal category $\mathbf{C}$, the monadic register $\mathbf{PSEF_C}$ for the category $\mathbf{Mon}(\mathbf{C})$ of monoids in $\mathbf{C}$ is defined as follows.

Signatures: A signature is a finitary, pointed strong endofunctor on $\mathbf{C}$.

Semantics: The abstract signature associated to any signature Σ is the forgetful functor $\Sigma\text{-}\mathbf{Mon} \to \mathbf{Mon}(\mathbf{C})$.

Validity proof: By Theorem 9.19. □

9.3. **Monoidal equational systems.** In this section, we combine the registers $\mathbf{ES_C}$ and $\mathbf{PSEF_C}$ of the two previous sections, yielding a new register $\mathbf{MES_C}$ obtained from $\mathbf{PSEF_C}$ by 'adding monoidal equations'.

Definition 9.23. Given finitary, pointed strong endofunctors Σ and Γ on a convenient monoidal category $\mathbf{C}$, a **monoidal functorial $\Sigma^{\circledast}$-term** of arity Γ is a parametric module morphism $\Gamma^{stmod} \to (\Sigma^{\circledast})^{monmod}$, where we recall Γ^{stmod} from Definition 9.16 and $(\Sigma^{\circledast})^{monmod}$ from Notation 9.20 and Example 2.15.

Definition 9.24. A **monoidal equational system** $\mathbb{E} = (\mathbf{C} \colon \Gamma \vdash L =_{\otimes} R : \Sigma)$ consists of

- a convenient monoidal category $\mathbf{C}$,
- two pointed strong endofunctors Σ and Γ on $\mathbf{C}$, and
- a parallel pair $L, R \colon \Gamma^{stmod} \rightrightarrows (\Sigma^{\circledast})^{monmod}$ of monoidal functorial $\Sigma^{\circledast}$-terms, which we call a **monoidal equation**.

Proposition 9.25. *Given any locally finitely presentable category* $\mathbf{C}$*, there are natural isomorphisms (in* Σ *and* Γ*) between*

(i) *monoidal functorial* $\Sigma^{\circledast}$*-terms of arity* Γ*, i.e., parametric module morphisms* $\Gamma^{stmod} \to (\Sigma^{\circledast})^{monmod}$*,*
(ii) *morphisms* $\Gamma^{\circledast} \to \Sigma^{\circledast}$ *of monads on* $\mathbf{Mon}(\mathbf{C})$*,*
(iii) *functors* $\Sigma\text{-}\mathbf{Mon} \to \Gamma\text{-}\mathbf{Mon}$ *over* $\mathbf{Mon}(\mathbf{C})$*,*
(iv) *morphisms* $(\Sigma + \Gamma)^{\circledast} \to \Sigma^{\circledast}$ *of monads on* $\mathbf{Mon}(\mathbf{C})$*, with section the canonical morphism* $\Sigma^{\circledast} \to (\Sigma + \Gamma)^{\circledast}$*,*
(v) *functors* $\Sigma\text{-}\mathbf{Mon} \to (\Sigma + \Gamma)\text{-}\mathbf{Mon}$ *over* $\mathbf{Mon}(\mathbf{C})$ *which are sections of the canonical functor* $(\Sigma + \Gamma)\text{-}\mathbf{Mon} \to \Sigma\text{-}\mathbf{Mon}$*, and*
(vi) *natural transformations* $\Gamma^{stmod} \circ U_{\Sigma} \to U_{\Sigma}$*.*

Proof. Both equivalences *(ii)* ⇔ *(iii)* and *(iv)* ⇔ *(v)* follow directly from Proposition 2.35 because by definition we have $\Sigma^{\circledast}\text{-alg} \cong \Sigma\text{-}\mathbf{Mon}$, and similarly for Γ and $\Sigma + \Gamma$. Furthermore, *(vi)* ⇔ *(iii)* and *(iii)* ⇔ *(v)* both follow by mere definition unfolding. This leaves us with proving that the first point agrees with one of the others.

For this, we exhibit a natural isomorphism

$$\frac{T\text{-alg} \to \Gamma\text{-}\mathbf{Mon}\,(\text{over } \mathbf{Mon}(\mathbf{C}))}{\Gamma^{stmod} \to T^{monmod}},$$

for any monad T on $\mathbf{Mon}(\mathbf{C})$. The result thus follows by taking $T = \Sigma^{\circledast}$.

- Given any $L\colon T\text{-alg} \to \Gamma\text{-}\mathbf{Mon}$ over $\mathbf{Mon}(\mathbf{C})$, consider the morphism $L^{\downarrow}\colon \Gamma^{stmod} \to T^{monmod}$ defined at any monoid M by
$$\Gamma(M) \xrightarrow{\Gamma(\eta_M)} \Gamma(T(M)) \xrightarrow{L(T(M))} T(M).$$
This indeed defines a morphism of the claimed type, by a simple diagram chasing, and furthermore this assignment is clearly natural in Γ and T.
- Conversely, given any $\alpha\colon U\Gamma \to VT$, let $\alpha^{\uparrow}\colon T\text{-alg} \to \Gamma\text{-}\mathbf{Mon}$ map any T-algebra structure $a\colon T(M) \to M$ on a monoid M to the Γ-algebra structure
$$\Gamma(M) \xrightarrow{\alpha_M} T(M) \xrightarrow{a} M.$$
Again, a simple diagram chase shows that this Γ-algebra structure satisfies the coherence law of Γ-monoids, and the assignment is natural in Γ and T.

These two maps are easily checked to be mutually inverse, thus proving the claim. □

Example 9.26. Recall from Example 9.15 the pointed strong endofunctor $\Sigma(X)(n) = X(n)^2 + X(n+1)$ for pure λ-calculus on $[\mathbf{Set}, \mathbf{Set}]_f$. As promised, let us now use this to complete the aborted treatment of the β-equation in Example 9.26, as we are now working with monoids, which we think of as objects equipped with substitution. We again take $\Gamma(X)(n) = X(n+1) \times X(n)$, and define L and R at any Σ-monoid T to map any $(f, e) \in T(n+1) \times T(n)$ to $\lambda(f)\ e$ and $f[e]$ respectively, where $f[e]$ denotes the result of substituting e for the $(n+1)$th variable of f. More precisely, $f[e] = \mu \circ Tu_e(f)$, where $u_e : n+1 \to Tn$ is $[\eta, e]$.

Let us now turn to defining algebras for a monoidal equational system.

Definition 9.27. For any monoidal equational system $\mathbb{E} = (\mathbf{C}\colon \Gamma \vdash L =_{\otimes} R : \Sigma)$, an **$\mathbb{E}$-algebra** is a monoid X whose $\Sigma^{\circledast}(X)$-algebra structure coequalises $L_X, R_X\colon \Gamma^{stmod}(X) \rightrightarrows (\Sigma^{\circledast})^{monmod}(X)$. An $\mathbb{E}$-algebra morphism is a morphism between underlying $\Sigma^{\circledast}$-algebras (a.k.a. Σ-monoids). We let $U_{\mathbb{E}}\colon \mathbb{E}\text{-alg} \to \mathbf{Mon}(\mathbf{C})$ denote the forgetful functor.

Mimicking Proposition 9.8, we now transfer this definition across the various correspondences of Proposition 9.25.

Proposition 9.28. *For any monoidal equational system $\mathbb{E} = (\mathbf{C}\colon \Gamma \vdash L = R : \Sigma)$ and $\Sigma^{\circledast}$-algebra $a\colon \Sigma^{\circledast}(X) \to X$, the following are equivalent:*

(a) *a coequalises $L_X, R_X\colon \Gamma^{stmod}(X) \to (\Sigma^{\circledast})^{monmod}(X)$,*
(b) *a coequalises the corresponding morphisms $\Gamma^{\circledast}(X) \to \Sigma^{\circledast}(X)$,*
(c) *the induced Σ-monoid structure $\Sigma^{stmod}(X) \to X$ belongs to the equaliser of the corresponding functors $\Sigma\text{-}\mathbf{Mon} \to \Gamma\text{-}\mathbf{Mon}$,*
(d) *the induced Σ-monoid structure $\Sigma^{stmod}(X) \to X$ belongs to the equaliser of the induced functors $\Sigma\text{-}\mathbf{Mon} \to \Gamma\text{-alg}$,*
(e) *a coequalises the corresponding morphisms $(\Sigma+\Gamma)^{\circledast}(X) \to \Sigma^{\circledast}(X)$,*
(f) *the induced Σ^{stmod}-algebra structure $\Sigma^{stmod}(X) \to X$ belongs to the equaliser of the corresponding functors $\Sigma\text{-}\mathbf{Mon} \to (\Sigma+\Gamma)\text{-}\mathbf{Mon}$,*
(g) *the corresponding natural transformations $\Gamma^{stmod} \circ U_{\Sigma} \to U_{\Sigma}$ have the same components at the induced Σ-algebra structure $\Sigma^{stmod}(X) \to X$.*

Proof.

- *(a)* $\Leftrightarrow$ *(c)* The corresponding functors Σ-**Mon** $\to$ Γ-**Mon** map the induced Σ^{stmod}-algebra to $\Gamma^{stmod}(X) \xrightarrow{K} (\Sigma^{\circledast})^{monmod}(X) \xrightarrow{a} X$, for $K = L, R$, so both sides unfold to the same thing.
- *(c)* $\Leftrightarrow$ *(d)* is clear.
- *(b)* $\Rightarrow$ *(a)* Follows from the fact that precomposing the induced morphisms
$$(\Gamma^{\circledast})^{monmod} \to (\Sigma^{\circledast})^{monmod}$$
by $\Gamma^{stmod} \to (\Gamma^{\circledast})^{monmod}$ yields the original L and R by construction.
- *(d)* $\Rightarrow$ *(b)* This holds because *(b)* is equivalent to equality of induced $\Gamma^{\circledast}$-algebra structures, which is further equivalent to equality of induced Γ-algebra structures.

The rest follows easily. □

Definition 9.29. For a given convenient monoidal category **C**, we define the monadic register $\mathbf{MES_C}$, called the **monoidal equational register**, as follows.

Signatures: A signature is a monoidal equational system.

Semantics: The abstract signature associated to a signature $\mathbb{E}$ is the forgetful functor $\mathbb{E}$-alg $\to$ **Mon**(**C**).

Validity proof: By Theorem 10.16 below. □

10. Computing initial algebras in the presence of equations

In this section, we establish the announced explicit descriptions of initial algebras, thereby proving that the registers introduced in §7 and §8 are valid. For this, we in passing also prove the validity of the registers from §9, and establish useful explicit descriptions of initial algebras for them too. In order, we characterise the underlying monad and initial algebra for suitable signatures in our registers featuring equations, namely the registers

- $\mathbf{ES_C}$ of equational systems (Definition 9.11),
- $\mathbf{MES_C}$ of monoidal equational systems (Definition 9.29),
- $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$ of equational modular signatures (Definition 7.15), and
- $\mathbf{Reg}^0[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ of equational facet-based signatures (Definition 8.11).

But before doing this, in §10.1, we study a well-known [4] refinement of finitariness, which we call **friendliness**, and prove the main foundational result about it. In §10.2–10.5, we then exploit this to characterise underlying monads and initial algebras for the announced registers.

10.1. **Reflexive coequalisers of friendly monoids.** Let us start from the announced result on monads (Theorem 7.22). The monads generated by our register $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$ with equations are, almost by definition, coequalisers in $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$. We have announced in Theorem 7.22 that their underlying functors are coequalisers in $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$. Technically, the goal is thus to delineate sufficient conditions for monad coequalisers $T_{\mathbf{E}} \rightrightarrows T_{\mathbf{D}} \twoheadrightarrow T'$ to be computed pointwise, in the sense that each $T_{\mathbf{E}}(X) \rightrightarrows T_{\mathbf{D}}(X) \twoheadrightarrow T'(X)$ is a coequaliser, and the monad structure is entirely determined by the family $(T'(X))_{X \in \mathbf{Set}^{\mathbb{P}}}$ (see [52, §V.3]). Roughly, this will work if both monads $T_{\mathbf{E}}$ and $T_{\mathbf{D}}$ preserve reflexive coequalisers. Because (finitary) monads are monoids in the category of (finitary) endofunctors, we can in fact generalise the result to monoids in a suitable category (see Proposition 10.17 below).

Preservation of reflexive coequalisers plays a fundamental role in the study of algebraic theories [4]. As we will use it a lot, let us give it a name.

Definition 10.1.

- A **reflexive pair** of morphisms is a pair $f, g\colon X \to Y$ of parallel morphisms, sharing a common section s, i.e., $s\colon Y \to X$ such that $fs = \mathrm{id}_Y = gs$.
- A **reflexive coequaliser** is a coequaliser of a reflexive pair.
- A functor is **friendly** when it is finitary and preserves reflexive coequalisers.

Remark 10.2. By [4, Theorem 7.7], when the considered categories are cocomplete, this is equivalent to preserving all **sifted** colimits.

Definition 10.3. An object X of a monoidal category is **⊗-friendly** (pronounced "tensor-friendly") when the functor $X \otimes -$ is friendly.

Proposition 10.4. *For any locally finitely presentable category* $\mathbf{C}$*, a finitary endofunctor on* $\mathbf{C}$ *is friendly iff it is ⊗-friendly as an object of* $[\mathbf{C}, \mathbf{C}]_f$ *(viewed as monoidal for the composition tensor product).*

Proof. Colimits of functors are computed pointwise [52, §V.4]. □

Notation 10.5. *By the proposition, in all of our use cases, ⊗-friendliness and friendliness are synonymous, hence we use the latter for simplicity.*

Definition 10.6. A monoidal category is **friendly** when it is convenient and all objects are friendly.

Example 10.7.

- The composition monoidal structure on finitary endofunctors on a locally finitely presentable category is convenient, though not friendly in general. By Proposition 10.4, the friendly objects are precisely the endofunctors preserving reflexive coequalisers.
- On categories of the form $\mathbf{Set}^{\mathbb{P}}$ for some set $\mathbb{P}$, though, by [4, Corollary 6.30], all finitary endofunctors are friendly, hence $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$ is friendly.

Proposition 10.8. *For any convenient monoidal category* $\mathbf{C}$*, the forgetful functor* $\mathbf{Mon}(\mathbf{C}) \to \mathbf{C}$ *creates reflexive coequalisers of friendly objects. More concretely, given a reflexive pair* $X \rightrightarrows Y$ *of monoid morphisms, if* X *and* Y *are friendly, then the coequaliser in* $\mathbf{C}$ *lifts uniquely to a cocone of monoids, which is again a coequaliser.*

Proof. Monoids are the algebras of the "free monoid" monad, which we denote by $(-)^\star$ in this proof. Furthermore, a forgetful functor from monad algebras always creates those colimits that the monad preserves (Proposition 2.23). It thus suffices to show that $(-)^\star$ preserves reflexive coequalisers of friendly objects.

Let thus $X_1 \rightrightarrows X_2 \twoheadrightarrow Z$ denote a reflexive coequaliser, with X_1 and X_2 friendly.

Let us first show that Z is again friendly. For this, we consider any reflexive coequaliser $A_1 \rightrightarrows A_2 \twoheadrightarrow C$. Then, we have:

$$\begin{array}{rcll} Z \otimes C & = & (\mathrm{colim}_i X_i) \otimes (\mathrm{colim}_j A_j) & \\ & \cong & \mathrm{colim}_i (X_i \otimes (\mathrm{colim}_j A_j)) & (\mathbf{C} \text{ is convenient}) \\ & \cong & \mathrm{colim}_i \mathrm{colim}_j (X_i \otimes A_j) & (\text{each } X_i \text{ is friendly}) \\ & \cong & \mathrm{colim}_j \mathrm{colim}_i (X_i \otimes A_j) & (\text{by interchange of colimits}) \\ & \cong & \mathrm{colim}_j ((\mathrm{colim}_i X_i) \otimes A_j) & (\mathbf{C} \text{ is convenient}) \\ & = & \mathrm{colim}_j (Z \otimes A_j), & \end{array}$$

as desired.

Furthermore, by Proposition 9.21 (with $\Sigma = \emptyset$), $Z^\star$ is the initial algebra of the endofunctor $H_Z = I + Z \otimes -$. Morevoer, H_Z is the coequaliser of $H_{X_1} \rightrightarrows H_{X_2}$, i.e., $(I + X_1 \otimes -) \rightrightarrows (I + X_2 \otimes -)$.

We then prove by induction that $H^n_{X_1}(0) \rightrightarrows H^n_{X_2}(0) \twoheadrightarrow H^n_Z(0)$ is again a (reflexive) coequaliser, for all $n \in \mathbb{N}$.

- The base case is trivial.
- Assuming that $H^n_{X_1}(0) \rightrightarrows H^n_{X_2}(0) \twoheadrightarrow H^n_Z(0)$ is a coequaliser, we consider the following diagram.

$$\begin{array}{ccccc}
H_{X_1}(H^n_{X_1}(0)) & \rightrightarrows & H_{X_1}(H^n_{X_2}(0)) & \longrightarrow & H_{X_1}(H^n_{\mathbf{Z}}(0)) \\
\downarrow\downarrow & & \downarrow\downarrow & & \downarrow\downarrow \\
H_{X_2}(H^n_{X_1}(0)) & \rightrightarrows & H_{X_2}(H^n_{X_2}(0)) & \longrightarrow & H_{X_2}(H^n_{\mathbf{Z}}(0)) \\
\downarrow & & \downarrow & & \downarrow \\
H_{\mathbf{Z}}(H^n_{X_1}(0)) & \rightrightarrows & H_{\mathbf{Z}}(H^n_{X_2}(0)) & \longrightarrow & H_{\mathbf{Z}}(H^n_{\mathbf{Z}}(0))
\end{array}$$

  By construction all columns are reflexive coequalisers, and by friendliness so are all rows. By [16, Lemma 8.4.2], the diagonal is thus again a (reflexive) coequaliser.

Finally, by interchange of colimits, we obtain that $X_1^\star \rightrightarrows X_2^\star \twoheadrightarrow Z^\star$ is also a coequaliser, as desired. □

Corollary 10.9. *Reflexive coequalisers of friendly monads on a finitely presentable category* $\mathbf{C}$ *are computed pointwise.*

10.2. **Initial algebras for equational systems.** In this section, we want to apply the previous corollary to characterise the induced monad and initial algebra for equational systems. For this, we should show that the monads $T_{\mathbf{D}}$ and $T_{\mathbf{E}}$ underlying the relevant parallel pair $\mathbf{D} \rightrightarrows \mathbf{E}$ in $\mathbf{Monadic}_f/\mathbf{C}$ are indeed friendly. What helps us here is that these monads are free on a friendly endofunctor, as we now show. We again state this in the abstract context of a convenient monoidal category.

Proposition 10.10. *In any convenient monoidal category, the free monoid on a friendly object is again friendly.*

Proof. We in fact prove the more general result that if $X \in \mathbf{C}$ preserves $\mathbf{D}$-colimits for a given category $\mathbf{D}$, in the sense that $X \otimes -$ preserves $\mathbf{D}$-colimits, then so does $X^\star$. By Corollary 2.24, it is enough to show that the forgetful functor from the category of algebras for the monad $X^\star \otimes -$ creates $\mathbf{D}$-colimits. But, by [46, Proposition 23.2], this category of algebras is isomorphic (over $\mathbf{C}$) to the category of algebras for the endofunctor $X \otimes -$. Thus, we are left with showing that the forgetful functor from this latter category creates $\mathbf{D}$-colimits, which follows from the next lemma. □

Lemma 10.11. *Let F be an endofunctor on a category $\mathbf{C}$. Then, the forgetful functor F-alg $\to \mathbf{C}$ creates any colimit that F preserves. More specifically, given a category $\mathbf{D}$ such that F preserves colimits of all diagrams $J\colon \mathbf{D} \to \mathbf{C}$, then the forgetful functor F-alg $\to \mathbf{C}$ creates colimits of all diagrams $J\colon \mathbf{D} \to F$-alg.*

Proof. Straightforward. □

Corollary 10.12. *The free monad on a friendly endofunctor on a finitely presentable category is friendly.*

We now want to apply Corollary 10.9 to prove a first free+quotient explicit description of initial algebras for equational systems. The exact same technique will be then applied to other registers in the following subsections, namely to monoidal equational systems, equational modular signatures, and equational facet-based signatures.

Before giving the explicit description, we need to introduce the following.

Definition 10.13. For any functorial term $K\colon \Gamma \to \Sigma^*$, let $\tilde{K}$ denote the monad morphism

$$(\Sigma + \Gamma)^* \cong \Sigma^* + \Gamma^* \xrightarrow{[\mathrm{id}_{\Sigma^*}, K']} \Sigma^*,$$

where $K'\colon \Gamma^* \to \Sigma^*$ denotes the monad morphism induced by K by freeness of Γ^*.

We may now state:

Theorem 10.14. *Let $\mathbb{E} = (\mathbf{C}\colon \Gamma \vdash L = R : \Sigma)$ be any equational system. Then:*

(i) *The forgetful functor $\mathbb{E}$-alg $\to \mathbf{C}$ is finitary monadic.*
(ii) *The finitary monad $\mathbb{E}^*$ underlying the forgetful functor $\mathbb{E}$-alg $\to \mathbf{C}$ is the coequaliser (in $\mathbf{Mnd}_f(\mathbf{C})$) of $\tilde{L}, \tilde{R}\colon (\Sigma + \Gamma)^* \rightrightarrows \Sigma^*$.*

Furthermore, if Σ and Γ are friendly (in particular when $\mathbf{C}$ is $\mathbf{Set}^{\mathbb{P}}$ for some set $\mathbb{P}$), we have:

(iii) *The above coequaliser $\mathbb{E}^*$ is created by the forgetful functor $\mathbf{Mnd}_f(\mathbf{C}) \to [\mathbf{C}, \mathbf{C}]_f$ (hence computed pointwise).*
(iv) *The initial $\mathbb{E}$-algebra is the coequaliser of*

$$\tilde{L}_0, \tilde{R}_0\colon (\Sigma + \Gamma)^*(0) \rightrightarrows \Sigma^*(0),$$

with its canonical Σ-algebra structure.

Proof. We start by expressing the category $\mathbb{E}$-alg as an equaliser in **CAT**.

For any functorial term $K\colon \Gamma \to \Sigma^*$, let $K'\colon \Sigma$-alg $\to \Gamma$-alg map any Σ-algbra $a\colon \Sigma(X) \to X$ to the composite

$$\Gamma(X) \xrightarrow{K_X} \Sigma^*(X) \xrightarrow{\tilde{a}} X,$$

where $\tilde{a}$ denotes the induced Σ^*-algebra structure on X.

The category $\mathbb{E}$-alg is then the equaliser of the (generally non-reflexive) pair below left.

$$\Sigma\text{-alg} \underset{R'}{\overset{L'}{\rightrightarrows}} \Gamma\text{-alg} \qquad\qquad \Sigma\text{-alg} \underset{\check{\check{R}}}{\overset{\check{\check{L}}}{\rightrightarrows}} (\Sigma + \Gamma)\text{-alg},$$

which already entails monadicity by Corollary 2.44. But it is also an equaliser of the reflexive pair above right. The rest thus follows from Corollary 10.9 using Corollary 10.12. □

Example 10.15. Let us recall Example 9.2, where we introduced an equational system whose algebras are sets equipped with an associative binary operation. Because we know how to compute coequalisers in sets, the theorem says that the free algebra on any $X \in \mathbf{Set}$ is obtained by quotienting the free Σ-algebra $\Sigma^*(X)$, obtained by freely adding a binary operation to X, under the following relation $\sim$. We first construct the free $(\Sigma+\Gamma)$-algebra $(\Sigma+\Gamma)^*(X)$ on X, obtained by freely adding a binary operation and a ternary operation, say f, to X. We then define two maps $L, R\colon (\Sigma+\Gamma)^*(X) \to \Sigma^*(X)$, by recursively interpreting $f(x,y,z)$ as $(x+y)+z$, resp. $x+(y+z)$. We finally define $\sim$ to be the smallest equivalence relation such that $x \sim y$ whenever $x = L(z)$ and $y = R(z)$ for some $z \in (\Sigma+\Gamma)^*(X)$.

10.3. **Initial algebras for monoidal equational systems.** In this section, we characterise the induced monad and initial algebra of monoidal equational systems, under mild additional hypotheses.

Theorem 10.16. *Let $\mathbb{E} = (\mathbf{C}\colon \Gamma \vdash L =_{\otimes} R : \Sigma)$ be any monoidal equational system. Then:*

(i) *The forgetful functor $\mathbb{E}$-alg $\to \mathbf{Mon}(\mathbf{C})$ is finitary monadic.*
(ii) *The finitary monad $\mathbb{E}^\star$ induced by the forgetful functor $\mathbb{E}$-alg $\to \mathbf{C}$ is the coequaliser of $L, R\colon (\Sigma+\Gamma)^\star \rightrightarrows \Sigma^\star$ in $\mathbf{Mnd}_f(\mathbf{C})$, where we recall that, for any finitary, pointed strong endofunctor F, $F^\star$ denotes the "free F-monoid" monad on $\mathbf{C}$.*

If $\mathbf{C}$, Σ, and Γ are friendly, then:

(iii) *The above coequaliser $\mathbb{E}^\star$ is created by the forgetful functor $\mathbf{Mnd}_f(\mathbf{C}) \to [\mathbf{C},\mathbf{C}]_f$, i.e., for any $X \in \mathbf{C}$, $\mathbb{E}^\star(X)$ is the coequaliser of $L_X, R_X\colon (\Sigma+\Gamma)^\star(X) \rightrightarrows \Sigma^\star(X)$, equipped with its canonical Σ-monoid structure.*
(iv) *The initial $\mathbb{E}$-algebra is the coequaliser of $L_0, R_0\colon (\Sigma+\Gamma)^*(\mathrm{id}) \rightrightarrows \Sigma^*(\mathrm{id})$, equipped with its canonical Σ-monoid structure.*

Proof. This will follow from Corollary 10.9 if we prove that both monads $(\Sigma+\Gamma)^\star$ and $\Sigma^\star$ are friendly (noting that the initial object is friendly in any convenient category). It is the purpose of the next lemma. □

Proposition 10.17. *For any finitary, pointed strong endofunctor F on a friendly monoidal category $\mathbf{C}$, if F is friendly, then so is the "free F-monoid" monad $F^\star$.*

Proof. This is a direct generalisation of the proof of Proposition 10.8 (which corresponds to the case $F = \emptyset$), using the fact that the free F-monoid monad maps X to the initial algebra of $A \mapsto I + X \otimes A + F(A)$, as recalled in Proposition 9.21. □

10.4. **Initial algebras for $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$.** In this section, we characterise the underlying monad and initial algebra of equational modular signatures, i.e., signatures of the register $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$, filling in the missing bits from §7.4. We first deal with the register $\mathbf{RegMnd}^0_f(\mathbf{Set}^{\mathbb{P}})$ without equations, by compiling (in the sense of Definition 5.8) to the register $\mathbf{PSEF}_{[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{P}}]_f}$ of pointed strong endofunctors on $[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{P}}]_f$. We then tackle the whole register $\mathbf{RegMnd}_f(\mathbf{Set}^{\mathbb{P}})$, using friendliness.

Let us first deal with the case without equations. Recalling from Definition 7.18 the endofunctor Σ_S on $[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{P}}]_f$ induced by a modular signature S, the idea in this case is that the assignment $S \mapsto \Sigma_S$ may be viewed as mapping signatures of $\mathbf{RegMnd}^0_f(\mathbf{Set}^{\mathbb{P}})$ to signatures of $\mathbf{PSEF}_{[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{P}}]_f}$, i.e., pointed strong endofunctors. We first establish this by

equipping Σ_S with a pointed strength, then use compilation to transport the problem, and conclude.

Proposition 10.18. *For any modular signature S, the endofunctor Σ_S admits a pointed strength given at any $P \in [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$, $Q \in \mathrm{id}/[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$, $X \in \mathbf{Set}^{\mathbb{P}}$, and operation of arity (d, p), say with $d = (F \circ \Theta)^{(p_1,\dots,p_n)}$, by applying $F(P(-)) \cdot \mathbf{y}_r$ to the obvious morphism $Q(X) + \sum_i \mathbf{y}_{p_i} \to Q(X + \sum_i \mathbf{y}_{p_i})$.*

Proof. Straightforward. □

Remark 10.19. In a bit more detail, letting $S = (d_i, p_i)_{i \in I}$, we have $\Sigma_S = \sum_{i \in I} \Sigma_{d_i, p_i}$. The pointed strength is defined as the coproduct

$$(\sum_{i \in I} \Sigma_{d_i,p_i})(P) \circ Q = \sum_{i \in I} (\Sigma_{d_i,p_i}(P) \circ Q) \xrightarrow{\sum_{i \in I} st^i_{P,Q}} \sum_{i \in I} \Sigma_{d_i,p_i}(P \circ Q),$$

where for all $i \in I$, say with $d_i \cong (F \circ \Theta)^{(q_1,\dots,q_n)}$ and $p_i \in \mathbb{P}$, $st^i_{P,Q,X}$ is the obvious morphism

$$F(P(Q(X) + \sum_{j \in n} \mathbf{y}_{q_j})) \cdot \mathbf{y}_{p_i} \to F(P(Q(X + \sum_{j \in n} \mathbf{y}_{q_j}))) \cdot \mathbf{y}_{p_i}.$$

Proposition 10.20. *The assignment $S \mapsto \Sigma_S$ defines a compilation*

$$\mathbf{RegMnd}^0_f(\mathbf{Set}^{\mathbb{P}}) \to \mathbf{PSEF}_{[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f}.$$

More concretely, for any modular signature $S = (d_i, r_i)_{i \in I}$, there exists an isomorphism

$$S\text{-alg} \cong \Sigma_S\text{-}\mathbf{Mon}$$

of categories over $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$.

Proof. The $\mathbf{Set}$-valued module morphisms $\rho_i \colon d_i(X) \to X_{r_i}$ correspond by adjunction (7.1) to $\mathbf{Set}^{\mathbb{P}}$-valued module morphisms $\tilde{\rho_i} \colon d_i(X) \cdot \mathbf{y}_{r_i} \to X$, hence by copairing to one Σ_S^{stmod}-algebra structure $\Sigma_S^{stmod}(X) \to X$ (recalling Definition 9.16), and thus to Σ_S-monoid structure on X. This correspondence extends straightforwardly to morphisms, hence defining the desired functor $S\text{-alg} \to \Sigma_S\text{-}\mathbf{Mon}$ over $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$. Since it is bijective, the functor is an isomorphism. □

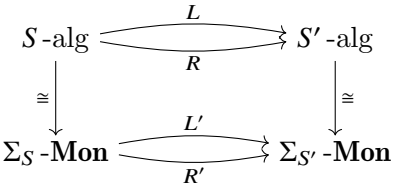
Corollary 10.21. *For any modular signature S, the forgetful functor $S\text{-alg} \to \mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$ is monadic, and furthermore the free S-algebra on an endofunctor $X \in [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$ is the free Σ_S-monoid $\Sigma_S^\star(X)$, as characterised in Proposition 9.21.*

In particular, $\Sigma_S^*(\mathrm{id})$ has a canonical S-algebra structure, which is initial.

We now want to characterise the initial E-algebra, for any equational modular signature $E = (S, S', L, R)$, where we recall that L and R are functors $S\text{-alg} \to S'\text{-alg}$ over $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$. Clearly:

Proposition 10.22. *For any equational modular signature $E = (S, S', L, R)$, $E\text{-alg}$ is the equaliser in $\mathbf{CAT}$ of L and R.*

By Proposition 10.20, L and R induce functors $L', R' \colon \Sigma_S\text{-}\mathbf{Mon} \to \Sigma_{S'}\text{-}\mathbf{Mon}$ over $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}})$ making the following square commute serially.



By Corollary 2.36, L' and R' induce a reflexive parallel pair of monad morphisms

$$L'', R'' \colon (\Sigma_S + \Sigma_{S'})^\star \rightrightarrows \Sigma_S^\star,$$

which is isomorphic to the pair

$$\tilde{L}, \tilde{R} \colon (S + S')^\star \rightrightarrows S^\star$$

from §7.4. The tuple $(\Sigma_S, \Sigma_{S+S'}, L'', R'')$ forms a monoidal equational system, say $\mathbb{E}_E$, over $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$, whose algebras are by construction isomorphic to (S, E)-algebras. This readily entails:

Corollary 10.23. *The assignment*

$$E = (S, S', L, R) \mapsto \mathbb{E}_E := (\Sigma_{S'} \vdash L'' =_\otimes R'' : \Sigma_S)$$

induces a compilation

$$\mathbf{RegMnd}_f\,(\mathbf{Set}^{\mathbb{P}}) \to \mathbf{MES}_{[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f}.$$

More concretely, for any equational modular signature E, we have an isomorphism

$$E\text{-alg} \cong \mathbb{E}_E\text{-}\mathbf{Mon}$$

of categories over $\mathbf{Mnd}_f\,(\mathbf{Set}^{\mathbb{P}})$.

As a result, we readily obtain by Theorem 10.16:

Corollary 10.24. *Let $E = (S, S', L, R)$ be any equational modular signature. Then:*

(i) *The forgetful functor E-alg $\to \mathbf{Mnd}_f\,(\mathbf{Set}^{\mathbb{P}})$ is monadic.*

(ii) *The finitary monad $E^\star$ underlying the forgetful functor*

$$E\text{-alg} \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$$

is the reflexive coequaliser in $\mathbf{Mnd}_f\,([\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f)$ of $L'', R'' \colon (\Sigma_S + \Sigma_{S'})^\star \rightrightarrows \Sigma_S^\star$.

(iii) *The above coequaliser is created by the forgetful functor*

$$\mathbf{Mnd}_f\,([\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f) \to [[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f, [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f]_f$$

(hence computed pointwise).

(iv) *The initial E-algebra is the coequaliser of $(L'')_0, (R'')_0 \colon (\Sigma_S + \Sigma_{S'})^*(\mathrm{id}) \rightrightarrows \Sigma_S^*(\mathrm{id})$, equipped with its canonical S-algebra structure.*

10.5. **Initial algebras for $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$.** In this section, we prove monadicity of the register $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ of equational facet-based signatures, and characterise underlying monads and initial algebras. For this, we proceed essentially as in the previous section, but more simply since the intricacies related to Σ-monoids do not arise. We are thus able to compile

- $\mathbf{Reg}^0[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ to the register $\mathbf{EF}_{[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f}$ of endofunctors on $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$, and
- $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ to the register $\mathbf{ES}_{[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f}$ of equational systems over $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$.

Recalling from Definition 8.17 the endofunctor Σ_S associated to any facet-based signature S, we have:

Proposition 10.25. *The assignment $S \mapsto \Sigma_S$ defines a compilation*

$$\mathbf{Reg}^0[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f \to \mathbf{EF}_{[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f}.$$

More concretely, for any facet-based signature $S = (d_i, s_i)_{i \in I}$, there exists an isomorphism

$$S\text{-alg} \cong \Sigma_S\text{-alg}$$

of categories over $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$.

Proof. As in the proof of Proposition 10.20, by the adjunction

$$[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f \underset{(-)_s}{\overset{(-)\cdot \mathbf{y}_s}{\rightleftarrows}} [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}]_f \,\bot,$$

the facet morphisms $\rho_i \colon d_i(X) \to X_{s_i}$ correspond to natural transformations $\tilde{\rho}_i \colon d_i(X) \cdot \mathbf{y}_{s_i} \to X$, hence by copairing to Σ_S-algebra structure $\Sigma_S(X) \to X$. This correspondence extends straightforwardly to morphisms, hence defining the desired functor $S\text{-alg} \to \Sigma_S\text{-alg}$ over $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$. Since it is bijective, the functor is an isomorphism. □

We now want to show that, for any equational facet-based signature E, the forgetful functor $E\text{-alg} \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ is monadic, and to explicitly characterise the corresponding monad and initial algebra. For this, we can exhibit $E\text{-alg}$ as an equaliser of finitary monadic functors over $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ and apply Corollary 2.44:

Proposition 10.26. *For any equational facet-based signature $E = (S, S', L, R)$, $E\text{-alg}$ is the equaliser in* **CAT** *of L and R.*

Proof. Straightforward. □

But in fact, perhaps more conveniently, we can also compile into equational systems. By Proposition 10.25, L and R induce functors $L', R' \colon \Sigma_S\text{-alg} \to \Sigma_{S'}\text{-alg}$ over $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ making the following square commute serially.

$$\begin{array}{ccc} S\text{-alg} & \underset{R}{\overset{L}{\rightrightarrows}} & S'\text{-alg} \\ \cong \downarrow & & \downarrow \cong \\ \Sigma_S\text{-alg} & \underset{R'}{\overset{L'}{\rightrightarrows}} & \Sigma_{S'}\text{-alg} \end{array}$$

This readily entails:

Corollary 10.27. *The assignment*

$$E = (S, S', L, R) \mapsto \mathbb{E}_E := (\Sigma_{S'} \vdash L' = R' : \Sigma_S)$$

induces a compilation

$$\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f \to \mathbf{ES}_{[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f}.$$

More concretely, for any equational facet-based signature $E = (S, S', L, R)$, we have isomorphisms

$$E\text{-alg} \cong \mathbb{E}_E\text{-alg}$$

of categories over $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$*, and*

$$E^\star \cong \mathbb{E}^\star_E$$

of finitary monads thereupon.

As a result, we readily obtain by Theorem 10.14:

Corollary 10.28. *Let* $E = (S, S', L, R)$ *be any equational facet-based signature. Then:*

(i) *The forgetful functor* E-alg $\to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ *is monadic.*
(ii) *The finitary monad* $E^\star$ *which underlies the forgetful functor* E-alg $\to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ *is the reflexive coequaliser in* $\mathbf{Mnd}_f([\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f)$ *of* $L', R' \colon (\Sigma_S + \Sigma_{S'})^* \rightrightarrows \Sigma_S^*$.
(iii) *The above coequaliser is created by the forgetful functor* $\mathbf{Mnd}_f([\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f) \to [[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f, [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f]_f$ *(hence computed pointwise).*
(iv) *The initial* E*-algebra is the coequaliser of* $(L')_0, (R')_0 \colon (\Sigma_S + \Sigma_{S'})^*(0) \rightrightarrows \Sigma_S^*(0)$*, equipped with its canonical* S*-algebra structure.*

11. Applications

In this section, we design a signature in $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$ for each of the announced languages. One exception is Positive GSOS systems: for them, we go further, by recasting them as the signatures of a subregister of $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$.

11.1. **The call-by-value, simply-typed, big-step λ-calculus.** Recall from §4.1, that the simply-typed, call-by-value, big-step λ-calculus forms a transition monad, where we take $\mathbb{P} = \mathbb{S}$ to be the set of types (generated from some fixed set of type constants). The monad T over $\mathbf{Set}^{\mathbb{P}}$ is given by values, the source state functor S_1 is given by application binary trees, and the second state functor S_2 is the identity.

Let us now design a signature for this transition monad. Let $F : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$ be specified by the signature of $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ consisting of two families of operations $\mathrm{app}_{t,t'} : \Theta_{t\to t'} \times \Theta_t \to \Theta_{t'}$ and $val_t : \mathbf{I}_t \to \Theta_t$. Our signature for call-by-value, simply-typed, big-step λ-calculus is presented in the following table

Monad and state functors	T	S_1	S_2
	$\lambda_{t,t'} : (F_{t'} \circ \Theta)^{(t)} \to \Theta_{t\to t'}$	F	Id
Rules	$\dfrac{}{val_t(v) \rightsquigarrow v}$		$\dfrac{e_1 \rightsquigarrow \lambda_{t,t'}(e_3) \qquad e_2 \rightsquigarrow w \qquad e_3[w] \rightsquigarrow v}{\mathrm{app}_{t,t'}(e_1, e_2) \rightsquigarrow v}$

where

- $-[-] : (S_1T)^{(t)}_{t'} \times T_t \to (S_1T)_{t'}$ denotes the substitution morphism;
- $S_1 = F$ and $S_2 = \mathrm{Id}$ are specified by easy signatures as in Remark 8.15;
- the rules should be understood as families of rules indexed by suitable types.

In a bit more detail, the first rule is indexed by the type t of v. The second one is indexed by two types t and t'. There are five metavariables, e_1, e_2, e_3, v, and w. We thus take $V := (S_1T)_{t\to t'} \times (S_1T)_t \times (S_1T)^{(t)}_{t'} \times T_{t'} \times T_t$.

11.2. **The $\bar{\lambda}\mu$-calculus.** Recall from §4.2 that the $\overline{\lambda}\mu$-calculus [39, 61] forms a transition monad with $\mathbb{P} = 2 = \{\mathbf{p}, \mathbf{s}\}$, where $\mathbf{p}$ stands for "programs" and $\mathbf{s}$ for "stacks". The placetaker monad T on $\mathbf{Set}^{\mathbb{P}}$ is given by programs and stacks. The set of transition types is $\mathbb{S} = 3 = \{\mathbf{c}, \mathbf{p}, \mathbf{s}\}$, where $\mathbf{c}$ stands for "commands", and both state functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$ are given by $S_1(A) = S_2(A) = (A_{\mathbf{p}} \times A_{\mathbf{s}}, A_{\mathbf{p}}, A_{\mathbf{s}})$. Let us repeat the grammar for the reader's convenience.

Commands	Programs	Stacks
$c ::= \langle e \mid \pi \rangle$	$e ::= x \mid \mu\alpha.c \mid \lambda x.e$	$\pi ::= \alpha \mid e \cdot \pi,$

Transitions are generated by the congruence rules and the following two rules

$$\langle \mu\alpha.c \mid \pi \rangle \to c[\alpha \mapsto \pi] \qquad \langle \lambda x.e \mid e' \cdot \pi \rangle \to \langle e[x \mapsto e'] \mid \pi \rangle.$$

Let us see how to specify this transition monad using our register.

We saw in Remark 8.15 that both state functors may be specified by operations

$$\langle - \mid - \rangle \colon \mathbf{I}_{\mathbf{p}} \times \mathbf{I}_{\mathbf{s}} \to \Theta_{\mathbf{c}} \qquad \eta_{\mathbf{p}} \colon \mathbf{I}_{\mathbf{p}} \to \Theta_{\mathbf{p}} \qquad \eta_{\mathbf{s}} \colon \mathbf{I}_{\mathbf{s}} \to \Theta_{\mathbf{s}}.$$

The monad is specified by operations

$$\mu : \Theta_{\mathbf{p}}^{(\mathbf{s})} \times \Theta_{\mathbf{s}}^{(\mathbf{s})} \to \Theta_{\mathbf{p}} \qquad \lambda : \Theta_{\mathbf{p}}^{(\mathbf{p})} \to \Theta_{\mathbf{p}} \qquad \cdot : \Theta_{\mathbf{p}} \times \Theta_{\mathbf{s}} \to \Theta_{\mathbf{s}},$$

with no equation.

The transition rules are almost as usual:

$$\frac{}{\langle \mu\langle e \mid \pi' \rangle \mid \pi \rangle \rightsquigarrow \langle e[\pi], \pi'[\pi] \rangle} \qquad \frac{}{\langle \lambda(e) \mid e' \cdot \pi \rangle \rightsquigarrow \langle e[e'] \mid \pi \rangle}$$

The first rule has metavariable module $V := T_{\mathbf{p}}^{(\mathbf{s})} \times T_{\mathbf{s}}^{(\mathbf{s})} \times T_{\mathbf{s}}$, the argument being (e, π', π). The second rule has $V := T_{\mathbf{p}}^{(\mathbf{p})} \times T_{\mathbf{p}} \times T_{\mathbf{s}}$. Let us finish by listing the various congruence rules.

$$\text{Commands:} \quad \frac{e \rightsquigarrow e'}{\langle e \mid \pi \rangle \rightsquigarrow \langle e' \mid \pi \rangle} \qquad \frac{\pi \rightsquigarrow \pi'}{\langle e \mid \pi \rangle \rightsquigarrow \langle e \mid \pi' \rangle}$$

$$\text{Programs:} \quad \frac{\langle e \mid \pi \rangle \rightsquigarrow \langle e' \mid \pi' \rangle}{\mu\langle e \mid \pi \rangle \rightsquigarrow \mu\langle e' \mid \pi' \rangle} \qquad \frac{e \rightsquigarrow e'}{\lambda(e) \rightsquigarrow \lambda(e')}$$

$$\text{Stacks:} \quad \frac{e \rightsquigarrow e'}{e \cdot \pi \rightsquigarrow e' \cdot \pi} \qquad \frac{\pi \rightsquigarrow \pi'}{e \cdot \pi \rightsquigarrow e \cdot \pi'}$$

11.3. **The π-calculus.** Recall from §4.3 that π-calculus [57] also forms a transition monad with $\mathbb{P} = 2 = \{\mathbf{c}, \mathbf{p}\}$ ($\mathbf{c}$ for "channels", $\mathbf{p}$ for "processes"). The placetaker monad is the identity on channels, and is defined on processes by the usual grammar

$$P, Q ::= 0 \mid (P \mid Q) \mid \nu a.P \mid \bar{a}\langle b \rangle.P \mid a(b).P,$$

where a and b range over channel names, and b is bound in $a(b).P$ and in $\nu b.P$. Processes are identified when related by the smallest context-closed equivalence relation $\equiv$ satisfying

$$0 \mid P \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad (\nu a.P) \mid Q \equiv \nu a.(P \mid Q),$$

where in the last equation a should not occur free in Q. Transitions are then given by the rules

$$\frac{}{\bar{a}\langle b \rangle.P \mid a(c).Q \longrightarrow P \mid (Q[c \mapsto b])} \qquad \frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R} \qquad \frac{P \longrightarrow Q}{\nu a.P \longrightarrow \nu a.Q}.$$

Let us recall that we denote any object $X \in \mathbf{Set}^{\mathbb{P}}$ by $X = (X_{\mathbf{c}}, X_{\mathbf{p}})$, so that we have $T(X) = (X_{\mathbf{c}}, T(X)_{\mathbf{p}}) \in \mathbf{Set}^{\mathbb{P}}$. Furthermore, transitions relate processes, so we have $\mathbb{S} = 1$ and $S_1(X) = S_2(X) = X_{\mathbf{p}}$.

Let us see how to specify this transition monad using our register. The state functor has been specified in Remark 8.15. The placetaker monad T is specified by operations

$0 : 1 \to \Theta_{\mathbf{p}} \quad | : \Theta_{\mathbf{p}} \times \Theta_{\mathbf{p}} \to \Theta_{\mathbf{p}} \quad \nu : \Theta_{\mathbf{p}}^{(\mathbf{c})} \to \Theta_{\mathbf{p}} \quad out : \Theta_{\mathbf{c}}^2 \times \Theta_{\mathbf{p}} \to \Theta_{\mathbf{p}} \quad in : \Theta_{\mathbf{c}} \times \Theta_{\mathbf{p}}^{(\mathbf{c})} \to \Theta_{\mathbf{p}}$

with equations

$$0|P \equiv P \quad P|Q \equiv Q|P \quad P|(Q|R) \equiv (P|Q)|R \quad \nu(P)|Q \equiv \nu(P|\mathbf{w}_{\mathbf{c}}(Q)),$$

almost copied verbatim from the standard presentation above, where $\mathbf{w}_{\mathbf{c}}(Q)$ denotes the action of $T(X) \to T(X + \mathbf{y}_{\mathbf{c}})$ on Q. Finally, the transition rules are

$$\frac{}{out(a,b,P)|in(a,Q) \rightsquigarrow P|(Q[b])} \qquad \frac{P \rightsquigarrow Q}{P|R \rightsquigarrow Q|R} \qquad \frac{P \rightsquigarrow Q}{\nu(P) \rightsquigarrow \nu(Q)}\ .$$

In particular, the third rule has as metavariable module $V := (\Theta_{\mathbf{p}}^{(\mathbf{c})})^2$.

Remark 11.1. We have accounted for the presentation of π-calculus through a reduction relation, but there is an important, alternative presentation based on a labelled transition system [57, Table 1.5 p38]. Let us explain why our framework cannot cover such a presentation. The problem is that it includes an input transition $a(x).P \xrightarrow{a(y)} P[x \mapsto y]$ in which y may be fresh. Letting T denote the monad for syntax as in §4.3, the correct way to model the fresh case is to take as metavariable object the T-module $T_{\mathbf{c}} \times T_{\mathbf{p}}^{(\mathbf{c})}$, whose elements are pairs of a channel a and a process P with a fresh variable. Because the module $T_{\mathbf{c}} \times T_{\mathbf{p}}^{(\mathbf{c})}$ is not free, this steps out of our register for transition monads.

11.4. **A register for Positive GSOS systems.** Finally, let us recall that Positive GSOS rules have the shape

$$\frac{x_i \xrightarrow{a_{i,j}} y_{i,j}}{op(x_1, \ldots, x_n) \xrightarrow{c} e},$$

where the variables x_i and $y_{i,j}$ are all distinct, $op \in O$ is an operation with arity n, and e is an expression potentially depending on all the variables.

We saw in §4.4 that each family of operations and rules yields a transition monad where

- $\mathbb{P} = 1$, because we are in an untyped setting,
- $\mathbb{S} = 1$ because states are terms,
- T denotes the term monad,
- $S_1(X) = X$, and
- $S_2(X) = \mathbb{A} \times X$, where $\mathbb{A}$ denotes the set of labels.

Let us now define a register $\mathbf{GSOS}^+$ for specifying positive GSOS systems [19]. This is a subregister of our register $\mathbf{RegTransMnd}_{\mathbb{P},\mathbb{S}}$, for untyped ($\mathbb{P} = \mathbb{S} = 1$) transition monads. Let us recall that signatures in this register consist of tuples $(\Sigma_T, \Sigma_1, \Sigma_2, \Sigma_{Trans})$, where $T = spec(\Sigma_T)$, $S_1 = spec(\Sigma_1)$, $S_2 = spec(\Sigma_2)$, and Σ_{Trans} is a signature in the register $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$.

In order to describe this subregister, we have to describe its class of signatures, and then assign to each such signature a tuple $(\Sigma_T, \Sigma_1, \Sigma_2, \Sigma_{Trans})$ as above.

A signature of the register $\mathbf{GSOS}^+$ consists of

- three sets O (for operations), $\mathbb{A}$ (for labels), and R (for rules),
- for each element o of O, a number m_o (the arity),
- for each rule,
  - an operation $o \in O$ (for the source of the conclusion),
  - a label $c \in \mathbb{A}$ (the label of the conclusion),
  - for each $i \leq m_o$,
    - a number n_i (the number of premises for this argument),
    - for each $j \leq n_i$, an element a_{ij} of $\mathbb{A}$ (for the label of the premise),
  - a term e in the syntax generated by O, potentially depending on $m_o + \sum_i n_i$ variables.

We now describe the tuple $(\Sigma_T, \Sigma_1, \Sigma_2, \Sigma_{Trans})$ associated to a signature as above:

- the signatures for both state functors have been given in Remark 8.15;
- the signature for the underlying monad is $\sum_{o \in O} \Theta^{m_o} \to \Theta$ (following §7.2);
- finally, each Positive GSOS rule yields a rule

$$\frac{\dots \qquad V_i \rightsquigarrow (a_{i,j}, V_{i,j}) \qquad \dots}{op_o(V_1, \dots, V_{m_o}) \rightsquigarrow (c, e)}$$

in our fibre signature Σ_{Trans} (in the register $\mathbf{RegTransStruct}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$).

11.5. **The differential λ-calculus.** Recall from §4.5 that the differential λ-calculus syntax is defined by

$$\begin{array}{rcll} e, f, g & ::= & x \mid \lambda x.e \mid e\langle U \rangle \mid De \cdot f & \text{(terms)} \\ U, V & ::= & 0 \mid e + U & \text{(multiterms)}, \end{array}$$

where terms and multiterms are considered equivalent up to the following equations.

$$e + e' + U = e' + e + U \qquad\qquad D(De \cdot f) \cdot g = D(De \cdot g) \cdot f.$$

Based on unary multiterm substitution $e[x \mapsto U]$ and partial derivation $\frac{\partial e}{\partial x} \cdot U$, the transition relation is defined as the smallest context-closed relation satisfying the rules below,

$$(\lambda x.e)\langle U \rangle \to e[x \mapsto U] \qquad\qquad D(\lambda x.e) \cdot f \to \lambda x. \left(\frac{\partial e}{\partial x} \cdot f \right)$$

where λ is linearly extended to multiterms: $\lambda x.(e_1 + \ldots + e_n)$ is notation for $\lambda x.e_1 + \ldots + \lambda x.e_n$.

We saw that this all forms a transition monad with one placetaker type and one transition type ($\mathbb{P} = \mathbb{S} = 1$).

Let us now design a signature specifying this transition monad. Such a signature consists of signatures for the state functors and monad, together with a signature for the transition module.

Signature for the placetaker monad Recalling that ! denote the finite multiset functor, the monad T of differential λ-calculus is specified by the signature S with operations

$$\lambda \colon \Theta^{(1)} \to \Theta \qquad -\langle - \rangle \colon \Theta \times !\Theta \to \Theta \qquad D- \cdot - \colon \Theta \times \Theta \to \Theta$$

and one equation:

$$D(De \cdot f) \cdot g \equiv D(De \cdot g) \cdot f. \tag{11.1}$$

By taking $!\Theta$ as type for the second argument of application, we directly identify multiterms as finite multisets, which explains why we do not need any further equation for enforcing order irrelevance of $+$.

Signature for the state functors Our state functors are $S_1 = \mathrm{Id}$ and $S_2 = !$, which are specified by easy signatures in the sense of Remark 8.15.

Signature for transitions Specifying the transition rules requires the following lemma.

Lemma 11.2. *There exist T-module morphisms*

$$\sigma\colon T^{(1)} \times !T \to !T \qquad \text{and} \qquad \delta : T^{(1)} \times !T \to !T^{(1)},$$

respectively called **unary multiterm substitution** *and* **partial derivation**, *satisfying the equations of* [61, Definition 6.3 and 6.4].

Proof sketch. By the explicit description of Corollary 10.24, T is a quotient of the initial Σ_S-monoid Z by Equation (11.1). We thus first define both operations from $Z^{(1)}$ by induction following [61, Definition 6.3 and 6.4], then check that both obtained morphisms coequalise the relevant parallel pair to extend them to morphisms from $T^{(1)}$. Finally, we check that both obtained morphisms are indeed module morphisms, which follows from [61, Lemma 6.10]. □

Now that both auxiliary operations σ and δ are defined, the main transition rules are

$$\frac{}{\lambda(t)\langle U\rangle \rightsquigarrow \sigma(t,U)} \qquad \frac{}{D(\lambda(t))\cdot u \rightsquigarrow \lambda(\delta(t,u))}\ ,$$

where we implicitly coerce $\lambda\colon T^{(1)} \to T$ into a morphism $!T^{(1)} \to !T$.

Furthermore, because reduction in the differential λ-calculus is context-closed, we need to include the following congruence rules, detailed in [61, Definition 6.18]:

$$\frac{t \rightsquigarrow U}{\lambda(t) \rightsquigarrow \lambda(U)} \qquad \frac{t \rightsquigarrow U}{Dt\cdot s \rightsquigarrow DU\cdot s} \qquad \frac{s \rightsquigarrow U}{Dt\cdot s \rightsquigarrow Dt\cdot U}$$

$$\frac{t \rightsquigarrow U}{t\langle S\rangle \rightsquigarrow U\langle S\rangle} \qquad \frac{t \rightsquigarrow U}{s\langle t+V\rangle \rightsquigarrow s\langle U+V\rangle}$$

where, as for λ above, we implicitly coerce some module morphisms to T into module morphisms to $!T$, also lifting some of their arguments from T to $!T$, namely we use

$$D(-)\cdot -\colon !T\times !T \to !T \qquad -\langle -\rangle\colon !T\times !T \to !T \qquad +\colon !T\times !T \to !T.$$

Remark 11.3.

- The first lifting is only used in cases where one of its arguments is a singleton multiset, i.e., we need $DU\cdot t$ and $Dt\cdot u$, but not $DU\cdot T$.
- In the second case, only the first argument needs lifting, i.e., we have $(e_1 + \ldots + e_n)\langle U\rangle = e_1\langle U\rangle + \ldots + e_n\langle U\rangle$.
- The last lifting is in fact mere multiset union.

12. Conclusion and perspectives

We have introduced transition monads as a generalisation of reduction monads, and demonstrated that they cover relevant new examples. We have introduced a register of signatures for specifying them. Let us briefly assess the scope of our register for transition monads, or more generally of the notion of transition monad itself.

All combinations of call-by-value vs. call-by-name, small-step vs. big-step, or simply-typed vs. untyped variants of λ-calculus should be handled smoothly.

Simple imperative languages like IMP [63, Chapter 2] may also be organised into transition monads, at least in a trivial way since, in the absence of first-class functions, reduction is generally defined on closed programs.

Languages whose transition rules involve some kind of evaluation contexts, like ML [54, Chapter 10], λ-calculi with `let rec` [13], or the substitution calculus [1], should also fit into the framework, although with a bit more work.

A first class of languages which clearly cannot be organised naturally as transition monads is those whose transitions involve non-free modules, as noticed in Remark 11.1. Extending the register $\mathbf{Reg}[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$ of §8 to cover such examples seems at hand.

Another, more problematic class of languages is those with advanced type systems, e.g., polymorphic or dependent types. Covering such examples is one of our longer-term goals.

Another limitation of our approach is the weakness of the induced induction principle. As discussed in §1, this is the price to pay for its simplicity. What is missing, in comparison with previous work like [9], is a kind of Grothendieck construction for signatures/registers. This works smoothly in most examples. However, in cases like the differential λ-calculus, this would require extending the definition of unary multiterm substitution and partial derivation (Lemma 11.2) to all models of the syntax. And this appears to leave some design choices open, which might be a reflection of the diversity of categorical semantics for differential λ-calculus [20, 26, 24, 45].

In the longer term, we plan to refine our register in a way ensuring that the generated transition system satisfies important properties like congruence of useful behavioural equivalences, confluence, or type soundness. In this direction, a result on congruence of applicative bisimilarity for a simpler register has recently been obtained by Borthelle et al. [23].

References

[1] Beniamino Accattoli. A fresh look at the lambda-calculus (invited talk). In Geuvers [35], pages 1:1–1:20. doi:10.4230/LIPIcs.FSCD.2019.1.

[2] Jirí Adámek, Stefan Milius, Nathan J. Bowler, and Paul Blain Levy. Coproducts of monads on set. In *Proc. 29th Symposium on Logic in Computer Science*, pages 45–54. IEEE, 2012. doi:10.1109/LICS.2012.16.

[3] Jiří Adámek and Jiří Rosicky. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994. doi:10.1017/CBO9780511600579.

[4] J. Adámek, J. Rosický, and E. M. Vitale. *Algebraic Theories: A Categorical Introduction to General Algebra*. Cambridge Tracts in Mathematics. Cambridge University Press, 2010. doi:10.1017/CBO9780511760754.

[5] Benedikt Ahrens. Initiality for typed syntax and semantics. *Journal of Formalized Reasoning*, 8(2):1–155, 2015. doi:10.6092/issn.1972-5787/4712.

[6] Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, 26:3–37, 2016. doi:10.1017/S0960129514000103.

[7] Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. High-level signatures and initial semantics. In Dan R. Ghica and Achim Jung, editors, *Proc. 27th EACSL Annual Conference on Computer Science Logic*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:22. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CSL.2018.4.

[8] Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular specification of monads through higher-order presentations. In Geuvers [35]. doi:10.4230/LIPIcs.FSCD.2019.6.

[9] Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Reduction monads and their signatures. *PACMPL*, 4(POPL):31:1–31:29, 2020. doi:10.1145/3371099.

[10] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:3)2015.

[11] Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. A categorical semantics for inductive-inductive definitions. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *Proc. 4th International Conference on Algebra and Coalgebra in Computer Science*, volume 6859 of *LNCS*, pages 70–84. Springer, 2011. doi:10.1007/978-3-642-22944-2_6.

[12] Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999. doi:10.1007/3-540-48168-0_32.

[13] Zena M. Ariola and Stefan Blom. Skew confluence and the lambda calculus with letrec. *Annals of Pure and Applied Logic*, 117(1-3):95–168, 2002. doi:10.1016/S0168-0072(01)00104-X.

[14] Nathanael Arkor and Marcelo Fiore. Algebraic models of simple type theories: A polynomial approach. In Hermanns et al. [40], pages 88–101. doi:10.1145/3373718.3394771.

[15] Michael Barr. Coequalizers and free triples. *Mathematische Zeitschrift*, 116:307–322, 1970.

[16] Michael Barr and Charles Wells. Toposes, triples, and theories. *Reprints in Theory and Applications of Categories*, 12, 2005. Originally published by: Springer, 1985.

[17] Françoise Bellegarde and James Hook. Subsitution: A formal methods case study using monads and transformations. *Sci. Comput. Program.*, 23(2-3):287–311, 1994. doi:10.1016/0167-6423(94)00022-0.

[18] Robert Oswald Blackwell, University of New South Wales. School of Mathematics, and University of New South Wales. Some existence theorems in the theory of doctrines, 1976. Thesis (Ph. D.)–University of New South Wales, 1976.

[19] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995. doi:10.1145/200836.200876.

[20] Richard Blute, J. Robin B. Cockett, and Robert A. G. Seely. Differential categories. *Mathematical Structures in Computer Science*, 16(6):1049–1083, 2006. doi:10.1017/S0960129506005676.

[21] Francis Borceux. *Handbook of Categorical Algebra*, volume 1 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994. doi:10.1017/CBO9780511525858.

[22] Francis Borceux. *Handbook of Categorical Algebra 2: Categories and Structures*. Cambridge University Press, 1994.

[23] Peio Borthelle, Tom Hirschowitz, and Ambroise Lafont. A cellular Howe theorem. In Hermanns et al. [40], pages 273–286. doi:10.1145/3373718.3394738.

[24] Thomas Ehrhard. An introduction to differential linear logic: proof-nets, models and antiderivatives. *Mathematical Structures in Computer Science*, 28(7):995–1060, 2018. doi:10.1017/S0960129516000372.

[25] Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1–3):1–41, 2003. doi:10.1016/S0304-3975(03)00392-X.

[26] Marcelo Fiore. Differential structure in models of multiplicative biadditive intuitionistic linear logic. In Simona Ronchi Della Rocca, editor, *Proc. 8th International Conference on Typed Lambda Calculi and Applications*, volume 4583 of *LNCS*, pages 163–177. Springer, 2007. doi:10.1007/978-3-540-73228-0_13.

[27] Marcelo Fiore. Second-order and dependently-sorted abstract syntax. In *Proc. 23rd Symposium on Logic in Computer Science*, pages 57–68. IEEE, 2008. doi:10.1109/LICS.2008.38.

[28] Marcelo Fiore and Chung-Kil Hur. On the construction of free algebras for equational systems. *Theoretical Computer Science*, 410(18):1704–1729, 2009. doi:10.1016/j.tcs.2008.12.052.

[29] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proc. 14th Symposium on Logic in Computer Science*, pages 193–202. IEEE, 1999. doi:10.1109/LICS.1999.782615.

[30] Marcelo Fiore and Philip Saville. List objects with algebraic structure. In Dale Miller, editor, *Proc. 2nd International Conference on Formal Structures for Computation and Deduction*, volume 84 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.FSCD.2017.16.

[31] Marcelo Fiore and Sam Staton. A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In *Proc. 21st Symposium on Logic in Computer Science*, pages 49–58. IEEE, 2006. doi:10.1109/LICS.2006.7.

[32] Marcelo Fiore and Daniele Turi. Semantics of name and value passing. In *Proc. 16th Symposium on Logic in Computer Science*, pages 93–104. IEEE, 2001. doi:10.1109/LICS.2001.932486.

[33] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *Proc. 14th Symposium on Logic in Computer Science* IEEE, 1999.

[34] Richard Garner. Combinatorial structure of type dependency. *CoRR*, abs/1402.6799, 2014. arXiv:1402.6799.

[35] Herman Geuvers, editor. *Proc. 4th International Conference on Formal Structures for Computation and Deduction*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.

[36] J.A. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology, IV: Data Structuring*, pages 80–144. Prentice-Hall, 1978.

[37] Makoto Hamana. Term rewriting with variable binding: An initial algebra approach. In *Proc. 5th International Conference on Principles and Practice of Declarative Programming* ACM, 2003. doi:10.1145/888251.888266.

[38] Makoto Hamana. Free Σ-monoids: A higher-order syntax with metavariables. In Wei-Ngan Chin, editor, *Proc. 2nd Asian Symposium on Programming Languages and Systems*, volume 3302 of *LNCS*, pages 348–363. Springer, 2004. doi:10.1007/978-3-540-30477-7_23.

[39] Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de lambda-termes et comme calcul de stratégies gagnantes.* PhD thesis, Paris Diderot University, France, 1995. URL: https://tel.archives-ouvertes.fr/tel-00382528.

[40] Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors. *Proc. 35th ACM/IEEE Symposium on Logic in Computer Science* ACM, 2020. doi:10.1145/3373718.

[41] André Hirschowitz, Tom Hirschowitz, and Ambroise Lafont. Modules over monads and operational semantics. volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:23. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.12.

[42] André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In *WoLLIC*, volume 4576 of *LNCS*, pages 218–237. Springer, 2007. doi:10.1007/3-540-44802-0_3.

[43] André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Information and Computation*, 208(5):545–564, 2010. doi:10.1016/j.ic.2009.07.003.

[44] Tom Hirschowitz. Cartesian closed 2-categories and permutation equivalence in higher-order rewriting. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:10)2013.

[45] Martin Hyland and Christine Tasson. The linear-non-linear substitution 2-monad. Submitted. URL: https://www.irif.fr/~tasson/doc/recherche/2020_hyland_tasson.pdf.

[46] G. M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22:1–83, 1980.

[47] G. Maxwell Kelly and A. John Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *Journal of Pure and Applied Algebra*, 89(1):163–179, 1993. doi:10.1016/0022-4049(93)90092-8.

[48] Joachim Kock. Notes on polynomial functors. Working notes, 2009. URL: http://mat.uab.es/~kock/cat/polynomial.pdf.

[49] Stephen Lack. On the monadicity of finitary monads. *Journal of Pure and Applied Algebra*, 140:65–73, 1997.

[50] Ambroise Lafont. *Signatures and models for syntax and operational semantics in the presence of variable binding*. PhD thesis, École Nationale Superieure Mines – Telecom Atlantique Bretagne Pays de la Loire – IMT Atlantique, 2019. arXiv:1910.09162.

[51] S. Lassen. *Higher Order Operational Techniques in Semantics*, chapter Relational reasoning about contexts, pages 91–135. Cambridge University Press, 1998.

[52] Saunders Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer, 2nd edition, 1998.

[53] Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO - Theor. Inf. and Applic.*, 33(6):507–534, 1999. doi:DOI:10.1051/ita:1999130.

[54] Benjamin C. Pierce, editor. *Advanced Topics in Types and Programming Languages*. MIT Press, 2004.

[55] John Power. Abstract syntax: Substitution and binders: Invited address. In Marcelo Fiore, editor, *Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS*

2007, New Orleans, LA, USA, April 11-14, 2007, volume 173 of *Electronic Notes in Theoretical Computer Science*, pages 3–16. Elsevier, 2007. doi:10.1016/j.entcs.2007.02.024.

[56] Jan Reiterman. A left adjoint construction related to free triples. *Journal of Pure and Applied Algebra*, 10:57–71, 1977. doi:10.1016/0022-4049(77)90028-7.

[57] Davide Sangiorgi and David Walker. *The π-calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[58] Gavin J. Seal. Tensors, monads and actions. *Theory and Applications of Categories*, 28(15):403–434, 2013.

[59] Sam Staton. General structural operational semantics through categorical logic. In *Proc. 23rd Symposium on Logic in Computer Science*, pages 166–177. IEEE, 2008. doi:10.1109/LICS.2008.43.

[60] Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proc. 12th Symposium on Logic in Computer Science*, pages 280–291, 1997. doi:10.1109/LICS.1997.614955.

[61] Lionel Vaux. *λ-calcul différentiel et logique classique : interactions calculatoires*. PhD thesis, Université Aix-Marseille 2, 2007.

[62] Mark Weber. Generic morphisms, parametric representations and weakly cartesian monads. *Theory and Applications of Categories*, 13:191–234, 2004.

[63] Glynn Winskel. *The formal semantics of programming languages - an introduction*. Foundation of computing series. MIT Press, 1993.